Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Systém pro podporu a hodnocení týmové práce studentů

The System for Supporting and Evaluating Students' Teamwork

2015 **Bc. Jiří Klusal**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jiří Klusal**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Systém pro podporu a hodnocení týmové práce studentů
The System for Supporting and Evaluating Students' Teamwork

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvoření systému, který bude nástrojem pro podporu týmové práce studentů, a to především se zaměřením na efektivní sledování a hodnocení jejich práce v rámci týmu. Součástí práce bude také návrh komplexní metodiky pro sledování, sběr, hodnocení a vizualizaci informací potřebných pro vyhodnocení práce jednotlivých studentů a týmů. Zaměření práce bude významně směřováno na týmy, jejich organizaci, strukturu, fungování, apod., stejně tak jako na vnímání tvrdých i měkkých dovedností při práci týmů a jednotlivců.

1. Nastudujte a popište základní principy projektového řízení, plánování a sledování výkonnosti týmu s ohledem na potřeby budovaného systému podle požadavků zadavatele.
2. Analyzujte možnosti využití týmové práce při výuce a navrhněte začlenění do systému.
3. Navrhněte a zpracujte metodiku pro měření, sledování a vyhodnocování práce studentů, a to v přímé vazbě na jejich týmovou organizaci.
4. Realizujte systém, který umožní týmovou práci studentu, a to včetně podpory navržené metodiky.
5. Zhodnoťte navržené řešení a jeho možnosti při nasazení nejen do výuky.

Seznam doporučené odborné literatury:

ARLOW, Jim; NEUSTADT, Ila. UML 2 a unifikovaný proces vývoje aplikací : objektově orientovaná analýza a návrh prakticky. 2., aktualiz. a dopl. vyd. Brno : Computer Press, 2007. 567 s. ISBN 978-802-5115-039.
DOLANSKÝ, Václav; MĚKOTA, Vladimír; NĚMEC, Vladimír. Projektový management. 1.vyd. Praha : Grada Publishing, 1996. 372 s. ISBN 80-716-9287-5.
SCHWALBE, Kathy. Řízení projektů v IT. Vyd. 1. Brno : Computer Press, 2007. 720 s. ISBN 978-802-5115-268.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Náplava**

Konzultant diplomové práce: Ing. Michal Radecký, Ph.D.

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *14. května 2015*

……..……
podpis studenta

Poděkování

Rád bych poděkoval Ing. Pavlu Náplavovi, Ing. Michalu Radeckému, Ph.D. a Ing. Karlu Mozdřeňovi za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Tato diplomová práce popisuje mnou navrhnutý a vytvořený nástroj pro podporu týmové práce studentů, a to se zaměřením na efektivnost sledování a hodnocení jejich práce v rámci týmu. Součástí této práce je teorie potřebná pro pochopení kontextu projektového řízení, procesů zpracovávání projektu, řízení zdrojů, hodnocení výkonu a vizualizace indikátorů výsledků práce. Tyto znalosti jsou dále aplikovány na studijní prostředí, kde učitelé a studenti nahrazují funkci projektových manažerů a projektového týmu. Další části práce popisuji návrh systému pro podporu týmové práce a následně jeho realizaci ve formě informačního systému. Součástí tohoto návrhu je i metodika pro sledování, sběr a autonomní hodnocení práce studentů. Výsledky mé práce experimentálně ověřuji a v poslední části práce hodnotím výsledky.

Klíčová slova

Projekt; tým; student; učitel; administrátor; úkol; událost, metodika, vyhodnocování, vizualizace; sledování.

Abstract

This diploma thesis describes analysis and creation of my system for supporting student's teamwork with focus on efficiency tracking and evaluation of their work inside the team. Part of the thesis is necessary theory, which is needed for understanding the context of project management, project processes, and human resources, evaluation of effectiveness and visualization of work results. This knowledge is then applied to the study environment, where teachers and students replace the functions of project managers and project team. In another part of the thesis I describe the design system for supporting student teamwork and their realization as an information system. A part of this design is methodology for tracking, gathering and autonomous evaluation of students work. At the end, I will verify and assess the results of my work.

Key words

Project; team; student; teacher; administrator; task; event; methodology; evaluation; visualization; tracking

Seznam použitých zkratek

Zkratka	Význam
POCO	Plain Old CLR Object
CLR	Common Language Runtime
DOM	Document Object Model
T4	Text Template Transformation Toolkit
SQL	Structured Query Language
RUP	Rational Unified Process
FTP	Protokol pro přenos souborů
HTTP	Hypertext Transfer Protocol

Seznam použitých termínů

Termín	Význam termínu
Features	Novinky (technologické)
Life cycle	Životní cyklus
Business plan	Podnikatelský plán ve smyslu zisku peněz
Learning management system	Systém podpory vyučování
Tracking issue system	Systém pro sledování úkolů (problémů)
Generic	Obecný
Role management	Spravování rolí
Deadline	Konečný termín
Parse	Vypiš
Extension metody	Rozšiřující metody
Request	Požadavek (nejčastěji http Get nebo Post)
Backlog	Seznam s chybami v rámci scrum metodiky
Framework	Platforma pro vývoj software
Commit	Procedura/Metoda uložení dat do databáze
SaveChanges	Metoda uložení dat do databázového kontextu
Lambda expression	Lambda výraz

Obsah

1 Úvod

V úvodní části bych rád zmínil můj první pracovní pohovor. V prvním kole si mě personalistka vyzpovídala ohledně technických dovedností a zjistila, co všechno umím. Druhé kolo se již skládalo z ústního pohovoru, kterému byl přítomen i projektový manažer a architekt systému.

Po úvodním přivítání, seznámení se s potencionálním systémem a náplní práce, mě architekt začal zkoušet. Mezitím projektový manažer sledoval mé reakce a odpovědi. To jsem však v té době nevěděl. Netušil jsem, že projektový manažer sleduje a zkoumá, co jsem za člověka, jaké jsou mé vlastnosti, jak se vyjadřuji a jestli bych „zapadl“ do kolektivu týmu.

Člověk by měl znát své místo, sám sebe dokázat ohodnotit a vědět, na co má předpoklady a na co ne. Ne vždy je toto tvrzení pravidlem. Tento poznatek je také jednou ze základních myšlenek mé práce.

V dnešní době se korporace skládají z jednotlivých oddělení, které zastřešují menší i větší projekty. Tyto projekty jsou tvořeny lidmi zařazenými na určité pozice, které jim zároveň předurčují jejich role v pracovním týmu. Při práci na zadaném projektu (úkolu) spolu musí lidé v týmu umět komunikovat, vědět, jakou roli v rámci týmu každý z nich představuje a spolupracovat. Proto je důležité také znát sebe sama, aby člověk věděl, co vlastně chce a co může dělat.

A kde jinde je k tomu nejlepší příležitost to zjistit? Při studiu. V průběhu předmětu, kde spolu tým řeší zadané úkoly, komunikuje, „brainstormuje“, nakonec i studenti mnohdy sami najdou odpověď na otázku (pokud ji již do té doby nenašli), čemu by se chtěli profesně věnovat a jaká role v rámci týmu by jim vlastně vyhovovala. Touto cestou se pak do budoucna můžou vydat.

V rámci této práce popíši základní principy fungování projektového řízení, plánování, metodiky vývoje a sledování výkonnosti týmu. Dále analyzuji požadavky a možnosti využití týmové práce při výuce s následným začleněním do systému. Navrhnu a zpracuji metodiku pro měření, sledování a vyhodnocování práce studentů. Realizuji systém, který umožní týmovou práci studentů, a to včetně podpory navržené metodiky. Na závěr zhodnotím navržené řešení, případnou rozšiřitelnost systému, a jeho možnosti nasazení do výuky jiných předmětů.

2 Teoretické základy

V této části popisuji teorii nezbytnou pro vypracování této práce. Uvádím, o čem je projektové řízení a jaké procesy probíhají na jeho pozadí. Definuji, co je to trojimperativ a o organizační struktury. Dále uvádím, jaké metodiky se používají při vývoji software, a nabízím možnosti, jak sledovat kvalitu práce. Na závěr doplňuji měkké a tvrdé dovednosti společně s osobnostním rozdělením.

2.1 Projektové řízení

Projektové řízení jako takové vychází především ze specifikace projektu [3]. Je nutné tedy nejprve definovat pojem projekt. Projekt je dočasná aktivita potřebná k vytvoření unikátního produktu, služby nebo obecně výsledku/výstupu. Má vymezený začátek i konec, má i vyhrazené zdroje. V rámci projektu máme také definované určité mezníky, dílčí cíle, které nám lépe pomáhají sledovat jeho postupnou realizaci. I pro dosažení těchto dílčích cílů je třeba využít všech dostupných znalostí, vlastností, dovedností, technik a efektivně je aplikovat. V tom nám pomáhá projektové řízení.

Projektové řízení se skládá z pěti fází:

- Koncept a zahájení projektu
- Definice a plánování
- Zahájení projektu
- Výkonost a řízení projektu
- Ukončení projektu

Lidé, kteří jsou zainteresování do projektového řízení, se musí starat o to, aby bylo efektivně dosaženo předpokládaných výsledků v závislosti na čase, rozsahu, cenně a také kvalitě. Základním kamenem projektového řízení je tzv. „projektový trojimperativ“.

Obrázek 2.1: *Trojimperativ* [29]

V ideálně vedeném projektu bychom měli dosáhnout určité shody mezi časem (Time), rozsahem (Scope) a cenou (Cost). Ne vždy tomu tak je. V praxi se, většinou z důvodu nedostatku finančních zdrojů, překročením časového harmonogramu (časového plánu), případně rozšířením požadavků klienta, kvalita konečného výsledku projektu od původního záměru liší.

Trojimperativ nám popisuje souvislosti, ale neříká nám, jak dosáhnout efektivity. K těmto účelům slouží nástroje, které dokáží usnadnit realizaci projektu vůči výše popsaným kategoriím.

K těm nejdůležitějším patří:

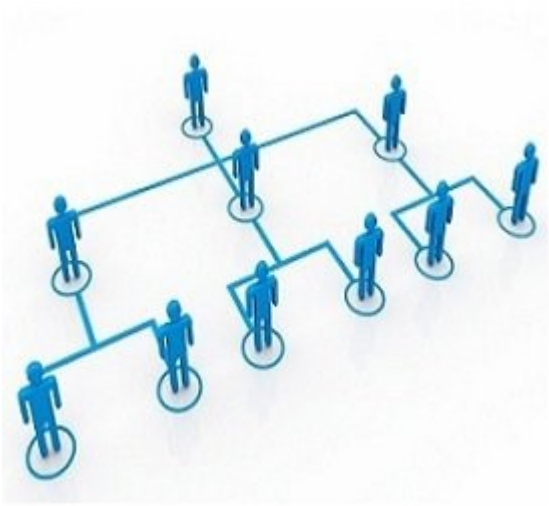
- Řízení rozsahu - definice cílů, analýza požadavků, řízení změn
- Řízení času - ganttovy diagramy, crash analýza, monitorování
- Řízení nákladů - analýza návratnosti, odhady nákladů
- Řízení kvality - metriky kvality, kontrolní diagramy, statistické metody
- Řízení lidských zdrojů - motivační techniky, matice zodpovědnosti, organizace projektu

2.2 Organizace a projektový tým

V praxi se projektové řízení zaměřuje na dosažení cílů prostřednictvím opakovaných činností. Ty souvisejí s plněním jednotlivých úkolů stejnými lidmi, kteří spolupracují v rámci jednoho týmu. Tyto týmy jsou již dnes nedílnou součástí především velkých firem a korporací. Dokáží totiž lépe reagovat na fluktuaci projektového trojimperativu a dynamicky alokovat větší

množství lidi na případné nedostatky. Fungují většinou napříč nebo ve shodě s organizační strukturou podniku.

Organizační struktury dělíme na dva základní typy: liniové a štábní [6]. Tyto dva typy se od sebe liší především mírou centralizace, pravomocemi, členitostmi podřízených, náplněmi, vztahy a případně i časem trvání. Pokud tyto základní typy zkombinujeme, dostaneme tzv. „kombinovanou“ organizační strukturu.



Obrázek 2.2: *Liniová organizační struktura* [30]

2.2.1 Liniová organizační struktura

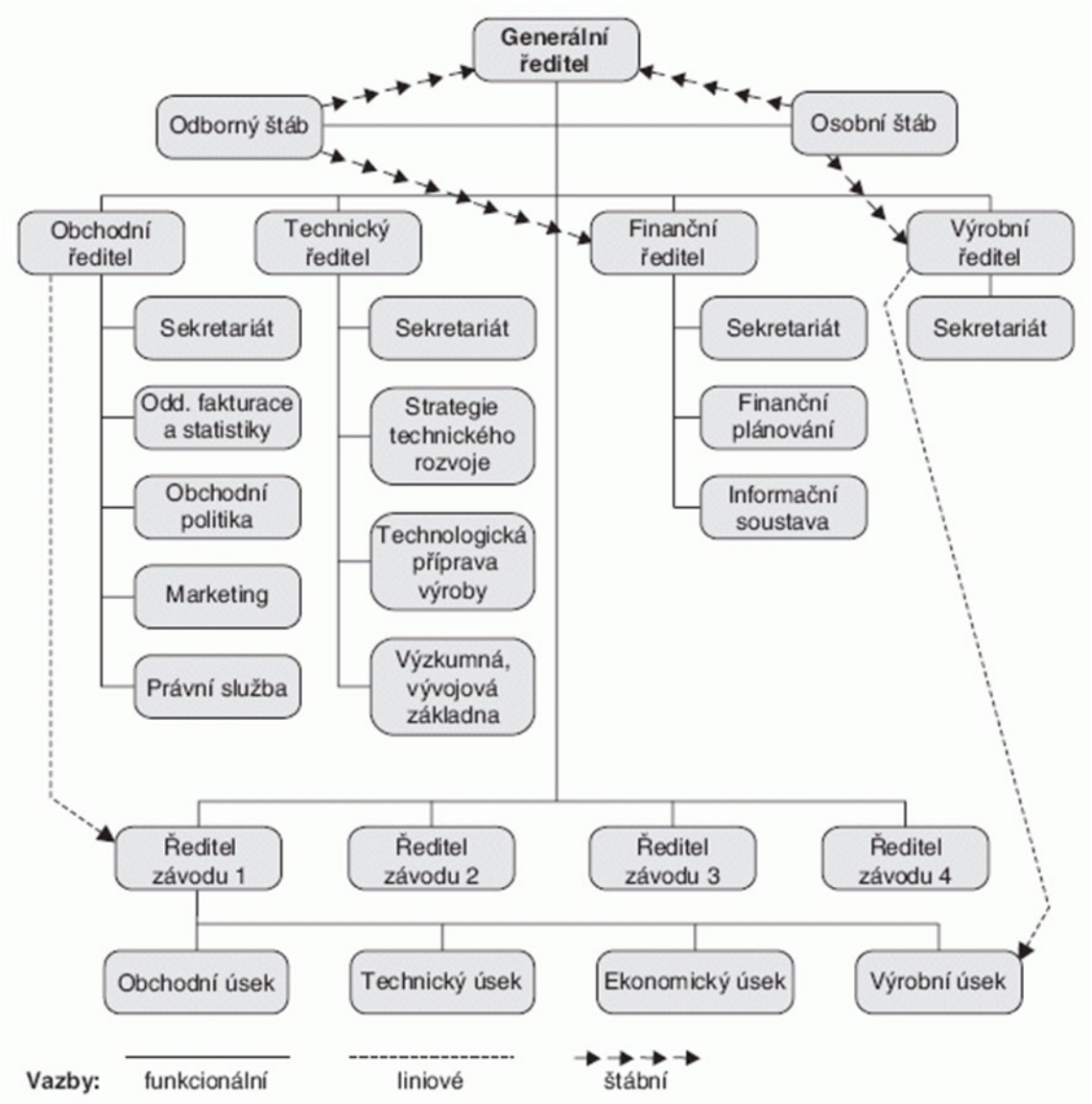
Liniovou organizační strukturu si můžeme představit jako malý podnik, který má v průměru okolo 40 zaměstnanců. Podnik má majitele, který řídí podnik a je na vrcholu řetězce. Pod ním jsou pak vedoucí/manažeři, kteří zodpovídají za jednotlivé zaměstnance.

2.2.2 Štábní organizační struktura

Štábní organizační struktura nemůže nikdy existovat samostatně, vždy je součástí nějaké jiné organizační struktury. Tedy například liniové nebo funkční. Štábní útvary plní funkci poradní, abychom dosáhli „správného“ rozhodnutí v rámci širšího celku. Takovýto útvar je tvořen experty a specialisty v daném oboru, jejichž úkolem je maximalizovat efektivitu jednotlivých organizačních struktur. V těch působí například sekretářka s majitelem firmy, a ta je vlastně sama o sobě štábním útvarem, který pomáhá majiteli firmy s tříděním formulářů.

2.2.3 **Kombinovaná organizační struktura**

Jak už název napovídá, jedná se kombinaci liniové a štábní organizační struktury. V hlavním principu jde o to, že například manažer (liniový pracovník) předá svoji práci (deleguje) na štábní jednotku.



Obrázek 2.3: *Kombinovaná organizační struktura* [6]

Mezi takovouto organizační strukturu patří také známá maticová organizační struktura, které vznikla sloučením funkční (např. front-end development, back-end development) a věcné dělby práce (např. projektu 1, projekt 2). Zaměstnanci tak odpovídají jednomu nebo více manažerům.

2.2.4 Projektový tým

Projektový tým tvoří osoby, které se součinně podílejí na zdárném dokončení projektu. Takovýto tým vzniká napříč kombinovanými strukturami a může být buď časově omezený, nebo i trvalý. Pak, v závislosti na složení lidí (který muže být plný expertů daného oboru), se může tým soustředit na řešení konkrétního problému/úkolu. Nebo dokáže řešit komplexní problémy, kdy složení týmu představuje experty z jiných odvětví oboru. Každý člen týmu má v rámci struktury týmu svou roli, ke které náleží pravomoci i povinnosti. Členové týmu jsou koordinováni vedoucím týmu tzv. Team Leaderem.

2.3 Metodiky vývoje

Trval projekt déle, než se předpokládalo? Překročil rozpočet hranici stanoveného minima na projekt hned několikrát? Byl zákazník nespokojen s tím, co jste mu dodali, a požadoval změnu, díky které byste mohli začít od znova? Tyto otázky si kladli vývojáři software již na začátku 90. let a snažili se nalézt řešení. Začaly vznikat nové formy metodik vývoje software, které byly pro firmy určitým způsobem zajímavé. Tyto formy metodik se především snažily předejít nejčastějším problémům vývoje software. Celosvětové výsledky pak poukázaly na to, že agilní metodiky opravdu fungují a jejich používání do určité míry zrychluje a zkvalitňuje vývoj software [4]. Kratší životní cyklus, inovace, zpětná vazba, pokrok, znalosti. To jsou hesla, kterými se řídí agilní metodiky.

2.3.1 Tradiční metodiky

Abychom lépe pochopili, proč vznikly agilní metodiky, tak si nejprve popíšeme tradiční metodiky vývoje životního cyklu software. Mezi představitele těchto metodik patří Vodopád, Spirálový model nebo také RUP. První problémy s těmito metodikami začaly vznikat u vývoje menších projektů, které vytvářely malé firmy. Tvorba software byla pomocí tradičních metodik velice pomalá a zdlouhavá. Metodika nutila malou skupinku lidí k nejrůznějším revizím, inspekcím a k dodržování přísných pravidel tvorby dokumentace projektu. Firma potřebovala velice rychle měnit požadavky zákazníků na systém. To však tradiční metodiky vývoje kvůli svému nekompromisnímu přístupu neumožňovaly. Tento světový problém vedl k zavedení agilních metodik.

2.3.2 Agilní metodiky

Slovíčko agilní nám už samo o sobě napovídá, že se bude jednat o velice „čilý, hbitý, pohyblivý“ přístup k tvorbě software [5]. Výhodou používání agilních metodik je, že dokáží velice rychle a flexibilně reagovat na změny požadavků. Tento přístup vývoje dokáže zajistit posun funkcionality systému v závislosti na čase a penězích. Zákazník tedy vždy dostane fungující software. Touto cestou, můžeme zákazníkovi v co nejkratším čase předvést fungující software a dostat od něj zpětnou vazbu. Vývoj tak probíhá v krátkých intervalech a zákazník může sledovat jeho vývoj.

Manifest agilního programování

Když se podíváme na jednotlivé agilní metodiky, které vznikaly před rokem 2001, zjistíme, že mají i přes své velké odlišnosti vždy něco společného. Tyto společné prvky byly v roce 2001 definovány v tzv. „Manifest agilního vývoje software“. Manifest uvádí v platnost několik pohledů na způsob vývoje software:

- Důraz na komunikaci mezi vývojovým týmem a zákazníkem
- Důraz na výslednou hodnotu pro zákazníka, před dokumenty a papírováním
- Spolupráce se zákazníkem, ještě než dojde ke sjednání samotného kontraktu
- Důraz na kvalitní a provozuschopný software s funkčností, která má obchodní hodnotu pro zákazníka
- Důraz na flexibilitu změn v přáních zákazníka

Hlavní principy

Nutnost vývojáře dnes a denně spolupracovat se zákazníkem. Požadavky zákazníka nelze hned přesně stanovit na začátku projektu. Zákazník na začátku nikdy přesně neví, co vlastně chce. Proto uděláme nejprve hrubý náčrtek zákazníkových požadavků a potřeb, a poté v průběhu životního cyklu vývoje software konzultujeme jednotlivé požadavky se zákazníkem.

Změnu požadavků může tedy provést zákazník například i v poslední fázi projektu. Díky krátkým iteračním dobám při vývoji lze nové požadavky zapracovat do projektu tak, abychom na konci dodali kvalitní a fungující software. Díky tomuto přístupu, může zákazník reagovat velice rychle na změnu trhu a dodávat vývojářům požadavky průběžně, tudíž jeho software bude mít opravdu přesně to, co potřebuje a co mu umožní konkurovat na trhu.

Zákazníka nezajímají diagramy, dokumentace návrhu, analýzy, testovací scénáře. Zákazník chce především vidět výsledky. Tedy to, zda software bude dodaný včas a zda bude opravdu umět, co ve skutečnosti chtěl a potřebuje. Toho docílíme jedině tak, že se mu budeme přizpůsobovat a zákazník bude postupně sledovat, jak plody jeho práce kvetou.

Agilní metodiky si kladou za cíl, aby software byl vyvinut pomocí co nejjednodušších postupů. Celkové řešení je pak mnohem pružnější a je připraveno na realizaci změn. Takto udělaný návrh řešení není ukončen jako samostatná etapa v životním cyklu software. Tento návrh je pod neustálým procesem iteračních změn.

Dokumentace není cílem projektu. Dokumentace v týmu slouží pouze jako prostředek k objasnění problému a k jeho lepšímu pochopení. V týmu se snažíme preferovat osobní komunikaci, která nám urychluje tvorbu software a šetří naše peníze. Dokonce ani dokumentace návrhu nám nezajistí úspěšné dokončení projektu. My potřebujeme mít pro zákazníka fungující produkt.

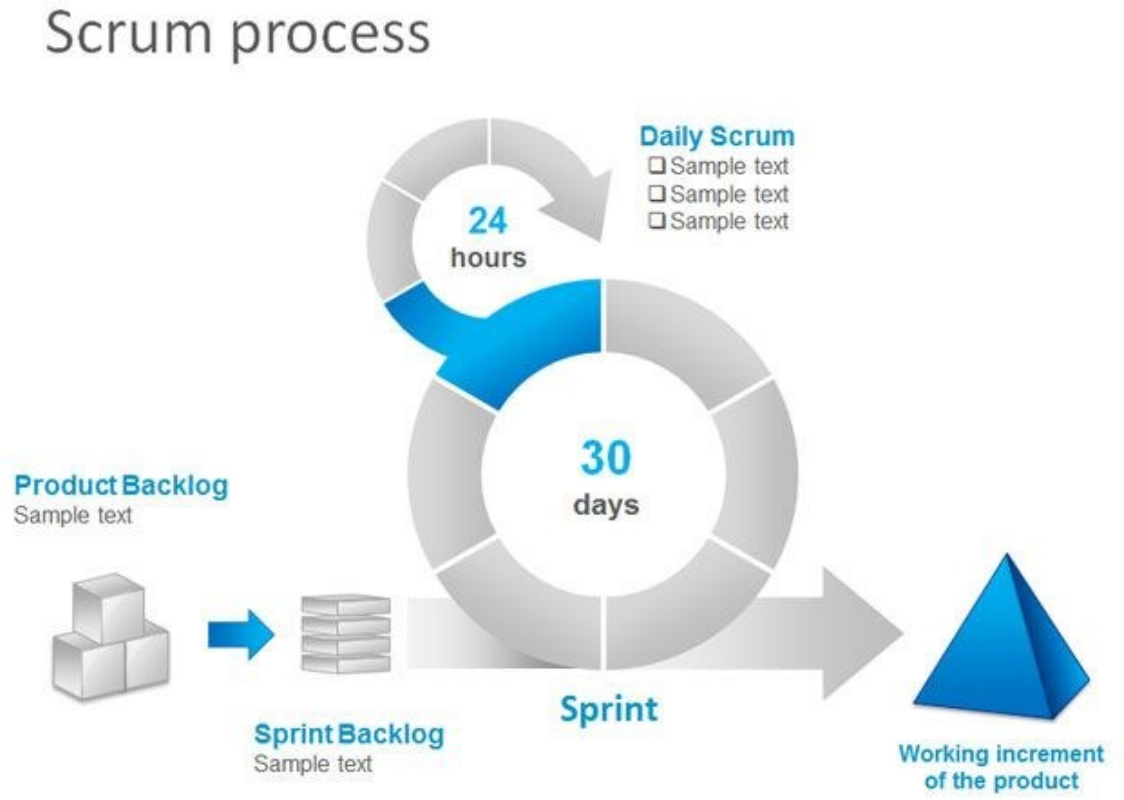
Práce v týmu a osobní komunikace v rámci týmů pomáhá lépe rozvinout řešené problémy a dojít tak k rychlému řešení. Výměna názorů, poznatků a informací podporuje zdravého ducha a zvyšuje produktivitu týmu.

Metodika SCRUM

SCRUM je zástupcem novodobých agilních metodik. Jedná se o rychlou a adaptivní metodiku, která je zaměřena na malé týmy. Slovo SCRUM pochází z rugby, kde do češtiny toto slovíčko překládáme jako „mlýn“. Tedy ve slangu rugby SCRUM znamená „jak dostat míč do hry“. Je zřejmé, že autoři chtěli zdůraznit význam strategie v týmu.

SCRUM je zaměřen na projektové řízení. Je koncipován tak, aby se dokázal velice flexibilně přizpůsobovat měnícím se požadavkům. Inspiroval se z již existujících praktik a soustředil se na řešení problémů, které při vývoji vznikaly. Tím metodika dokázala splňovat téměř všechny požadavky, které si klade manifest agilního programování.

Ve světě metodiky SCRUM se používají pojmy, které bych zde rád zmínil. Jedním z klíčových pojmů této metodiky je Sprint. Jedná se o typ iterace, který trvá do 4 týdnů. Vývoj software trvá přibližně 2-7 sprintů, přičemž záleží na množině užitných vlastností, která je dodána na konci každého Sprintu.



Obrázek 2.4: *Proces Scrumu* [31]

Důležitou součástí metodiky jsou SCRUM meetingy. Jde o princip každodenních malých projektových porad, při kterých se lidé v týmu snaží řešit problémy, které nastaly při vývoji a jsou ideálním prostředkem pro koordinaci projektové strategie. Členové týmu mají označení jako „pigs“, ostatní „chickens“. „Chickens“ na meetingu nesmí nic říkat. Na meetingu

musí každý účastník zodpovědět tři otázky: Co udělal od poslední schůzky? Co bude řešit do příštího setkání? Na jaké překážky narazil nebo jaké problémy vidí při řešení budoucího úkolu.

V rámci projektu vznikají tzv. dva Backlogy. Product Backlog obsahuje seznam požadavků, které očekáváme ve funkční aplikaci. Zákazník má tyto požadavky uspořádané podle priorit a logických celků, které potřebuje. Vývojový tým pak vezme určité požadavky, a ty implementuje. Tyto implementované požadavky jsou zaneseny do tzv. Sprint Backlogu. Sprint backlog je ve vlastnictví týmu a každý z členů týmu si vybírá, který požadavek bude dělat.

Figuruje zde několik rolí, přičemž každá role má svá práva a povinnosti:

- Scrum Master - sleduje, aby vývoj probíhal podle standardu, pravidel a praktik. Je tu především pro to, aby odstraňoval vzniklé překážky
- Product owner – komunikuje se zákazníkem a má na starosti product backlog
- Zákazník – sestavuje Product backlog
- Scrum tým – má na starosti sprint backlog a jeho plnění
- Management – spravuje administrativní záležitosti a schvaluje konečná rozhodnutí

Celý proces metodiky je rozdělen do čtyř fází:

- Plánovací fáze – soupis požadavků a plánování
- Vynášecí fáze – upřesnění požadavků do backlogů
- Fáze vývoje – průběh iterace, výše zmíněný Sprint
- Fáze dodávky – dodání produktu zákazníkovi

2.4 Matice zodpovědnosti

V rámci projektového řízení je celkem běžnou praxí, že projektový manažer vytváří tzv. matici zodpovědnosti. Jedná se o matici, kde jednotlivým osobám (například v rámci týmu), jsou přiřazeny dílčí části projektu, za které jsou zodpovědné. Z takovéto matice je pak zřejmé, kdo je za co zodpovědný. Maticí dokážeme i definovat jednotlivé role v rámci projektu.

Akce	Projektový manažer	Linde-gas.a.s	Škoda Plzeň	DPP
Projekt	x			
Vyřízení dotace	x			
Jednání s partnery	x			x
Vyřízení stavebních povolení	x	x		
Vodíkového autobus	x		x	
Čerpací stanice	x	x		
Kolaudace	x	x		
Zavážka plynu		x		x
Proškolení personálu	x	x		
Zkušební jízdy				x

Obrázek 2.5:*Matice zodpovědnosti* [32]

Jedná se o velmi jednoduchý, rychlý a přehledný způsob, jak v rámci týmu ihned vědět, kdo, co a kde dělá. Matice slouží nejen pro potřeby týmu, ale i pro osoby, které potřebují znát potřebné informace a patří mimo tým.

2.5 Myers-Briggs Type Indicator a Belbin

Nebo také MBTI osobnostní test [10]. Jedná se o test, který je hojně používán při určování osobnostních preferencí. Tyto testy se využívají především na pohovorech nebo potřebujeme-li zjistit vlastní preference. V projektovém řízení, konkrétně při přiřazování lidí do správných rolí/pozic, můžeme tohoto testu využít. Zjistíme tak, zdali určitý člověk zapadne do týmu. Test se zaměřuje na zkoumání čtyř dimenzí:

- Introvert/extrovert
- Smysly/intuice (ve smyslu získávání informací)
- Myšlení/cítění (způsob uvažování)
- Usuzování/vnímání (postoj daného člověka nebo také životní styl)

Výborná kombinace MBTI testu, je pak ve spojení s Belbin testem [28], v kterém Dr. Belbin definoval celkem 9 rolí (inovátor, vyhledávač zdrojů, realizátor, analytik, týmový pracovník, dotahovač, expert, koordinátor, usměrňovač), které představují role v týmu. Pomocí jeho testu pak určí, do které role člověk zapadne.

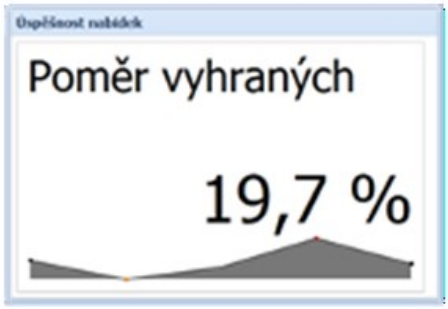
2.6 Kontrola kvality

Zlepšovat se? Být efektivnější? To jsou snad cíle všech. Ale my to potřebujeme nějakým způsobem měřit a vyhodnocovat. Sledování jakosti jako takové je především zaměřeno na produkty a služby. K tomu, abychom dosáhli co nejkvalitnějšího výrobku, nám obecně slouží normy. Jakmile je tedy výrobek hotový, zkontrolujeme, zdali odpovídá určitým parametrům/specifikacím. V tuto chvíli můžeme říci, zdali došlo ke shodě s požadavky. Ne vždy to však stačí. Klíčové je neustále se zlepšovat. A k tomu, abychom se posunuli dále, nám slouží metody řízení kvality - například velice známá six sigma [12]. Mezi další nástroje, které nám pomáhají, patří různé analytické techniky jako je například Ishikawův diagram (diagram příčin a následků), kde se snažíme najít všechny potencionální problémy, nebo metoda WIBI (Would I Buy It?). Řízení kvality obsahuje tři hlavní procesy:

- Plánování kvality - zajištění norem, standardů, metrik, aj.
- Zajištění kvality - kontinuální kontrola procesů
- Kontrola kvality - konkrétní kontrola výsledného produktu

2.7 Klíčové ukazatele výkonnosti

Nebo také KPI (Key Performance Indicator) [11] jsou ukazatele přiřazené určité službě, procesu, výrobku a mají za následek vyjádření určité kvality, výkonu nebo efektivity.



Obrázek 2.6: *Úspěšnost zakoupených nabídek* [11]

Jako příklad bych mohl uvést úspěšnost nabídek. Zde počet nabídek ukazuje úspěšnost zakoupení výrobku zákazníkem.

2.8 Benchmarking

Jedná se o metodu, která je založená na porovnávání výše zmíněných identifikátorů. Metoda vezme jednotlivé ukazatele a porovnává je s referenčními hodnotami jiných firem, týmů (pokud zde dochází k relevantnosti), případně s historickými ukazateli. Benchmarking neřekne, zdali je něco špatně nebo dobře, pouze poukazuje na rozdíly, které mohou být dále zkoumány, aby byla odhalena jejich příčina.

2.9 „Soft skills“ a „Hard skills“

Soft skills [13] a hard skills je anglický termín, který se užívá i v Česku pro měkké a tvrdé vlastnosti/dovednosti jednotlivců.

Hard skills jsou vlastnosti, které se dají vcelku dobře změřit a vyhodnotit [14]. Jedná se tedy o soubor vlastností, na které dokážeme aplikovat určitá pravidla, jako je například IQ stupnice, tedy inteligence, nebo jazyková dovednost (jazykové certifikáty), případně dosažené technické vzdělání (Ing. v oblasti informatiky a výpočetní techniky). Jsou to tedy znalosti/vědomosti, které se můžeme naučit.

Soft skills jsou vlastnosti, které jsou hůře měřitelné. Jedná se o skupinu vlastností jako je komunikace, porozumění, vedení, flexibilita, zodpovědnost, přístup, aj., které pokud k nim nemáme genetické předpoklady, si musíme osvojit a získat tréninkem.

Každá z těchto vlastností je důležitá, pro určitou roli.

3 Analýza začlenění týmu a jednotlivců do navrhovaného systému

Při popisu principiálního fungování se budu snažit postupovat strukturovaně. Nejprve si srovnáme learningové a tracking issue systémy, které bychom mohli potencionálně využít. Definujeme si funkční a nefunkční požadavky. Popíšeme si projekt v rámci výuky předmětu. Zároveň během svého výkladu budu modelovat příklad projektu, na kterém ukáži, jak systém funguje. Při popisu bude zřejmé, jak metodika hodnocení funguje.

3.1 Existující systémy

Na dnešním trhu se vyskytuje velké množství learning management systémů a tracking issue systémů. Většina z nich se snaží být ve velké míře generickými, aby obsloužily co největší množství zákazníků. Ve své podstatě se jedná o marketingový přístup, který firma preferuje. Problém nastává, když potřebujeme systém přizpůsobit vlastním potřebám. Při srovnávání požadavků na potřebný systém jsem bral v úvahu především následující systémy:

Obrázek 3.1: *Moodle logo* [18]

Moodle

Nebo také modular object-oriented dynamic learning environment [18]. Jedná se o open source learning systém, který nabízí možnost vytváření e-learningových kurzů. Moodle je velice flexibilní nástroj s velkým množstvím balíčků, díky kterým si uživatel dokáže nakonfigurovat prostředí tak, jak potřebuje. Systém nabízí možnost vytváření studijních kurzů, sdílení materiálů, publikování online testů nebo také propojení s wiki modulem. Což znamená připojení wiki systému k moodle a využití vlastní úpravy stránek.

TFS

Team Foundation Server je systém podporující issue tracking, source code managment, reporting a velké množství služeb pro podporu vývoje [20]. TFS je pro uživatele přístupné ve dvou variantách a to z webového prohlížeče, a pak dále z vývojových prostředí jako je Visual Studio nebo Eclipse. V rámci issue trackingu dokážeme vytvářet úkoly (user stories, tasky, bugy), které mají své stavy, aby účastníci systému věděli, co se děje.

Edmodo

Systém edmodo [19] se plně zaměřuje na výuku a sledování studentů, tedy další ze zástupců learningových systémů. Jedná se o sociální síť, která běží na podobném marketingovém modelu jako je facebook. V edomodu můžeme vytvářet skupiny, testy, úkoly, události, komunikovat aj. Je zda také možnost kontroly výuky ze strany rodičů, mohou kontrolovat známky a práci svých dětí.

Shrnutí

Jak jsem uvedl, na trhu je velké množství systémů, stačí si vybrat. Jako první možnost se nabízel Moodle. Je velice flexibilní a nabízí možnosti rozšíření základního nastavení o balíčky (moduly). Moodle je opravdu robustní systém, zejména když se jedná o konfigurování. Jeho velkou nevýhodou je skutečnost, že neobsahuje issue tracking v rámci týmů a role managmentu.

TFS prostředí je nabízené v rámci smluvních podmínek pro školy a tudíž i potencionálně použitelný systém pro naše potřeby. I když TFS nabízí issue tracking, tak celkově je systém uzpůsoben pro vývoj software.

Systém Edmodo nabízí funkcionalitu asi nejvíce vyhovující našim potřebám, ale jedná se o sociální síť. Veškerá data jsou uložena na serverech třetích stran a rozšiřitelnost aplikace nebo případná konfigurace jsou prakticky nemožné.

Abych celkově shrnul hlavní důvod, proč vytvořit vlastní systém pro podporu výuky je ten, že lze účinně spojit e-learning systém s issue trackingem, a to přímo na míru studijním potřebám. Tedy část, kdy učitelé dokáží vytvářet cvičení a týmy. V rámci jednotlivých týmů pak mohou členům přiřazovat role, hodnotit práci, srovnávat a zadávat úkoly. Tyto úkoly je možné sledovat a mají i své stavy. Tím, že si takovýto systém postavíme, dokážeme rychle a lehce reagovat na případná rozšíření, namísto toho, abychom upravovali TFS či moodle.

3.2 Požadavky na systém

3.2.1 Funkční požadavky

Požadavky jsou rozděleny podle rolí, tedy jakou funkcionalitu bude mít. Správou v požadavcích je myšlen CRUD. Tedy například "Správa událostí" pro administrátora je vytvoření, čtení, editace a smazání událostí.

Administrátorovi

- Správu uživatelů (přes web rozhraní, csv, web service)
- Správu semestru
- Správa cvičení
- Správa rozvrhu
- Správa úvodní stránky
- Správa událostí
- Správa souborů
- Správa týmů

Učiteli

- Správu projektů v rámci cvičení
- Správa dokumentů v rámci cvičení
- Přidávání úkolů v dané třídě hodnocení
- Správa týmu v rámci cvičení
- Hodnocení a srovnávání úkolů, týmů
- Správa událostí

Leader

- Správa nástěnky
- Přidávání úkolů v dané třídě hodnocení
- Správa týmových úkolů

Studentovi

- Upsat se na vypsaný úkol
- Přispívat do diskuze (komunikace v rámci týmu)
- Odevzdávat úkoly
- Nahrávat soubory

Uživatel

- Změnit osobní údaje

Systém

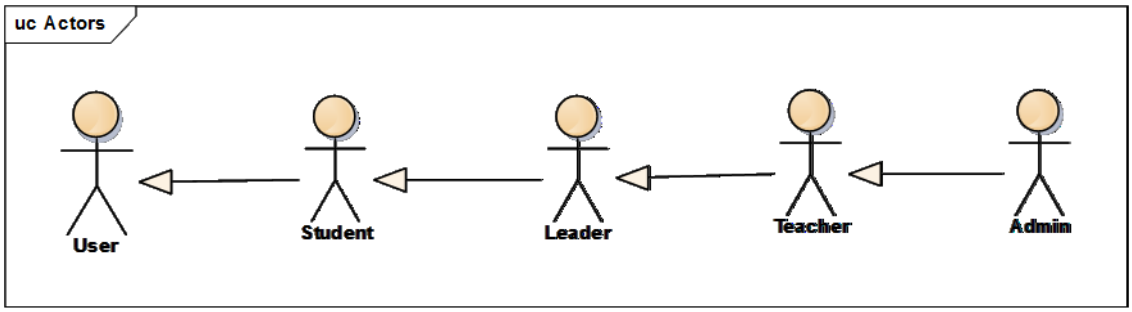
- Autentizace uživatelů
- Vyhodnotit a vizualizovat kvalitu práce

3.2.2 Nefunkční požadavky

- Systém bude vyžadovat pro svůj chod Microsoft Windows Server, na kterém poběží IIS
- Databázi Microsoft SQL Server
- Systém bude psán v jazyce C# a využívat technologií ASP MVC

3.3 Uživatelé (Aktéři) systému

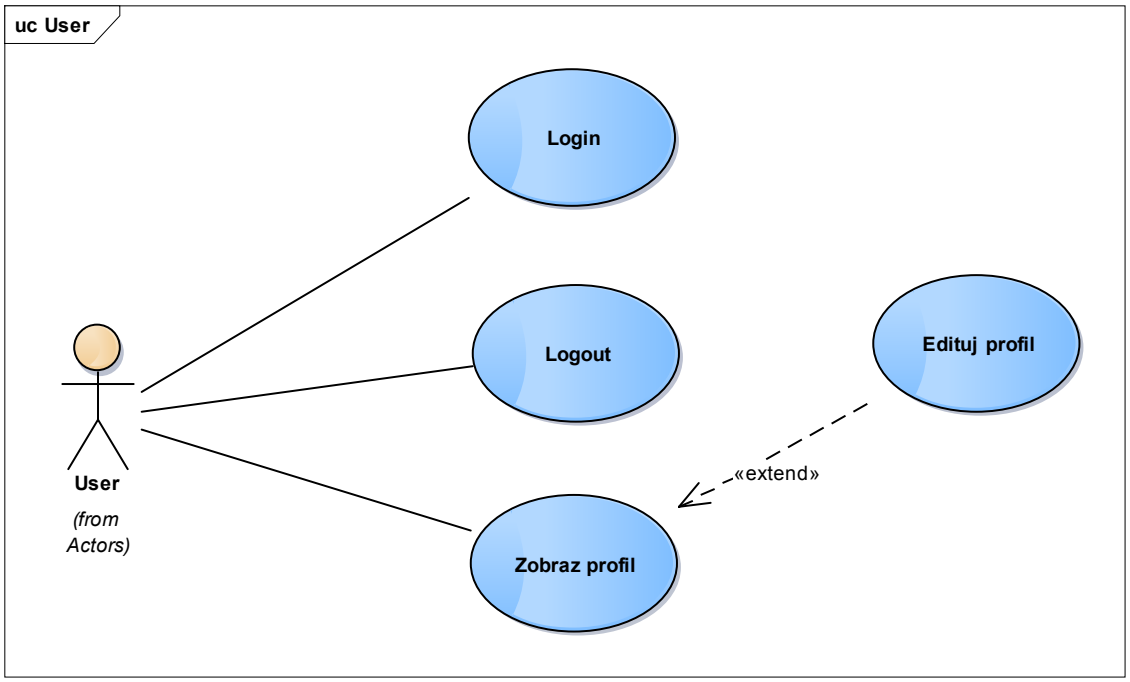
V systému existuje celkem pět aktérů. Administrátor (Admin), Učitel (Teacher), Leader, Student a Uživatel (User). Aktéři systému dědí práva, jak je zobrazeno na obrázku 3.2. V našem případě tento model hierarchie dědění práv je plně dostačující. Pokud bychom tedy postupovali zespod, tedy od uživatele, směrem nahoru, tak role směrem nahoru mají vždy o nějakou funkcionalitu navíc.



Obrázek *3.2*: *Use Case - Hierarchie uživatelů*

3.3.1 Uživatel

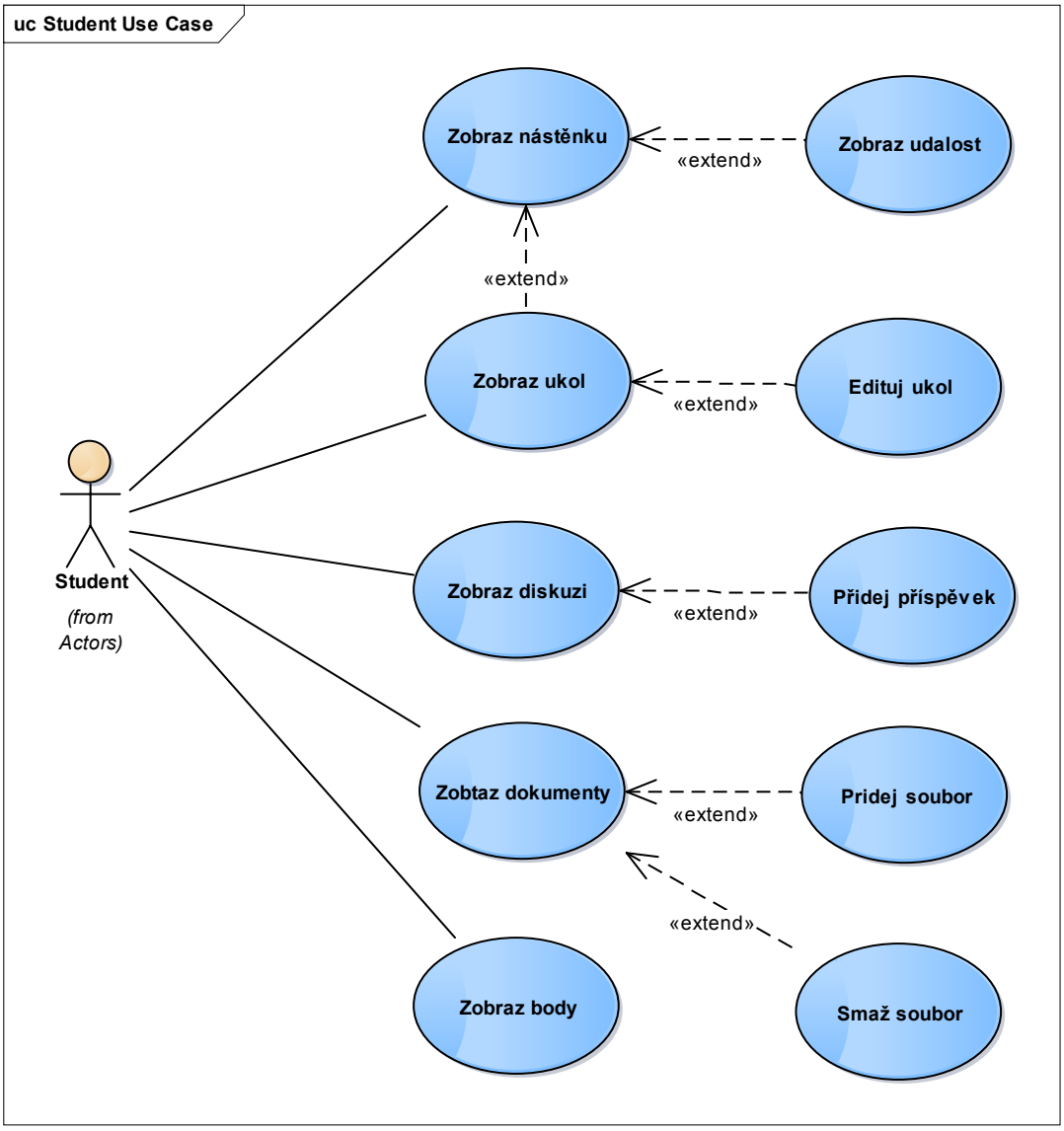
Uživatel je osoba, která byla přidána do systému s výchozí rolí User. Takto přidaná osoba (například ze souboru, webové služby nebo z webového rozhraní) si po nalogování do systému dokáže pouze změnit osobní údaje.



Obrázek 3.*3*:*Use Case - User(uživatel)*

3.3.2 Student

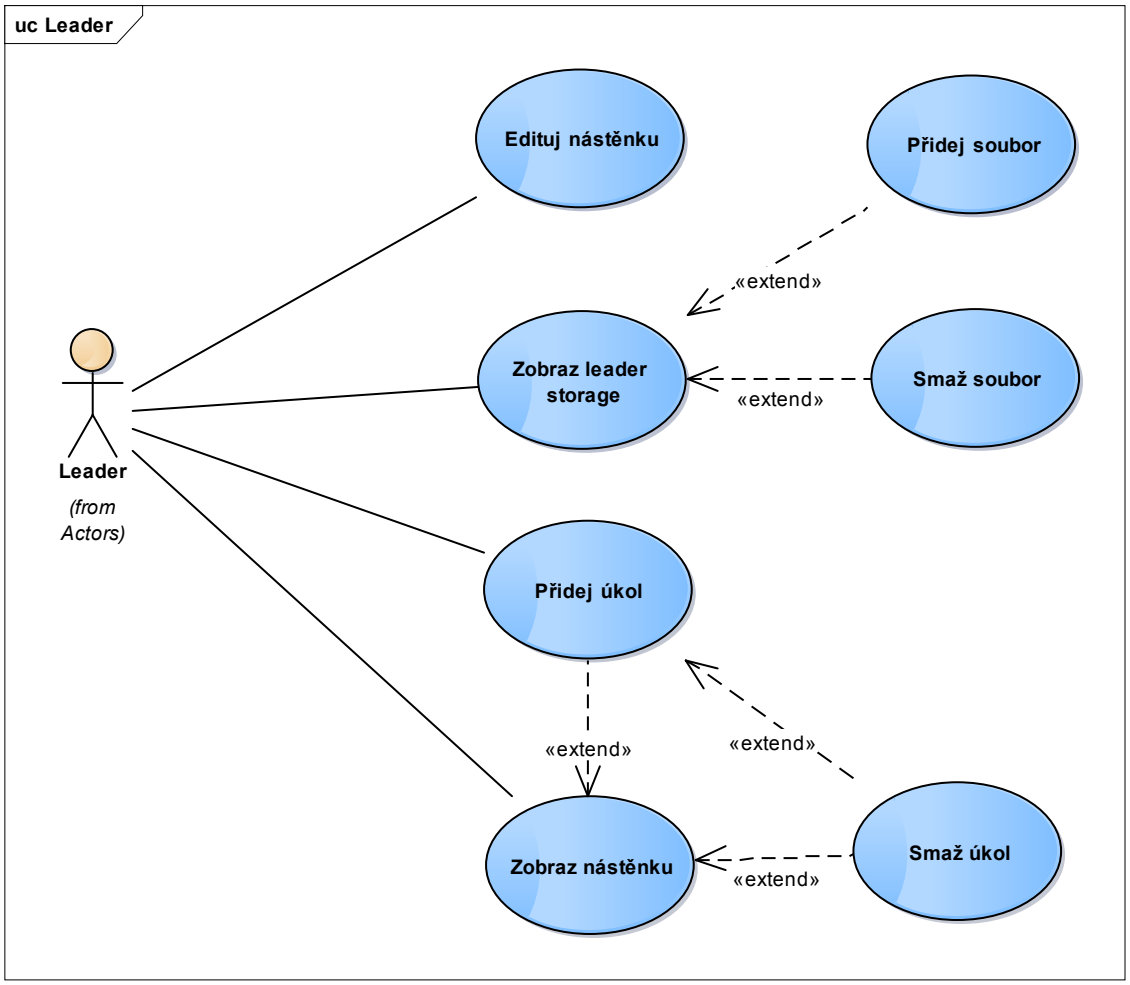
Student je osoba, která navštěvuje daný předmět v určitém semestru pod vybraným cvičením. Je součástí týmu a náleží mu určitá role, tedy i určitá funkcionalita. Z tohoto důvodu tuto osobu přiřadíme do role Student.



Obrázek 3.*4*: *Use Case - Student*

3.3.3 Leader

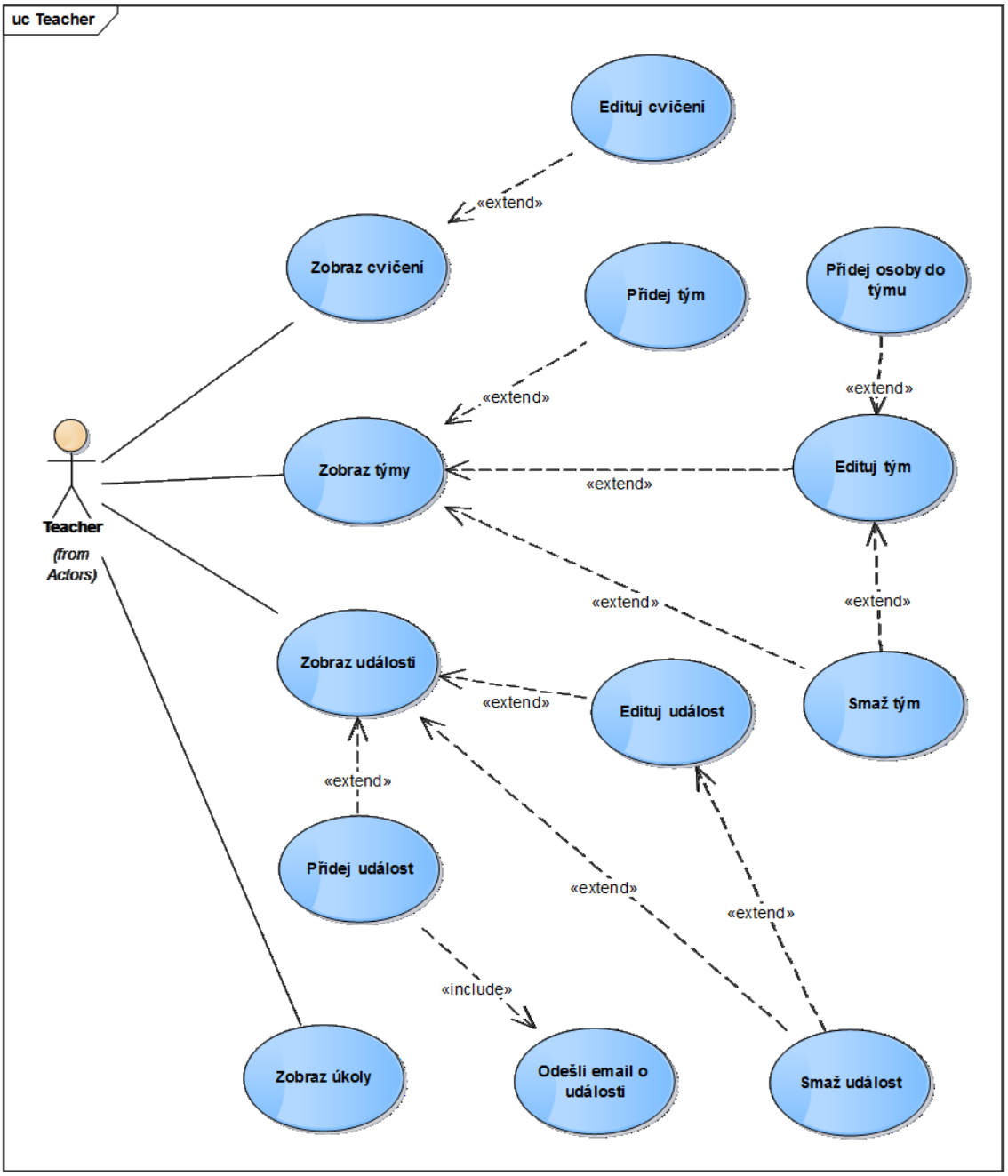
Roli Leadera představuje Studenta, který byl vybrán (ať už učitelem nebo členy týmu) aby reprezentoval tým. Má tedy i rozšířená práva oproti roli Student.



Obrázek 3.5: *Use Case - Leader*

3.3.4 Učitel

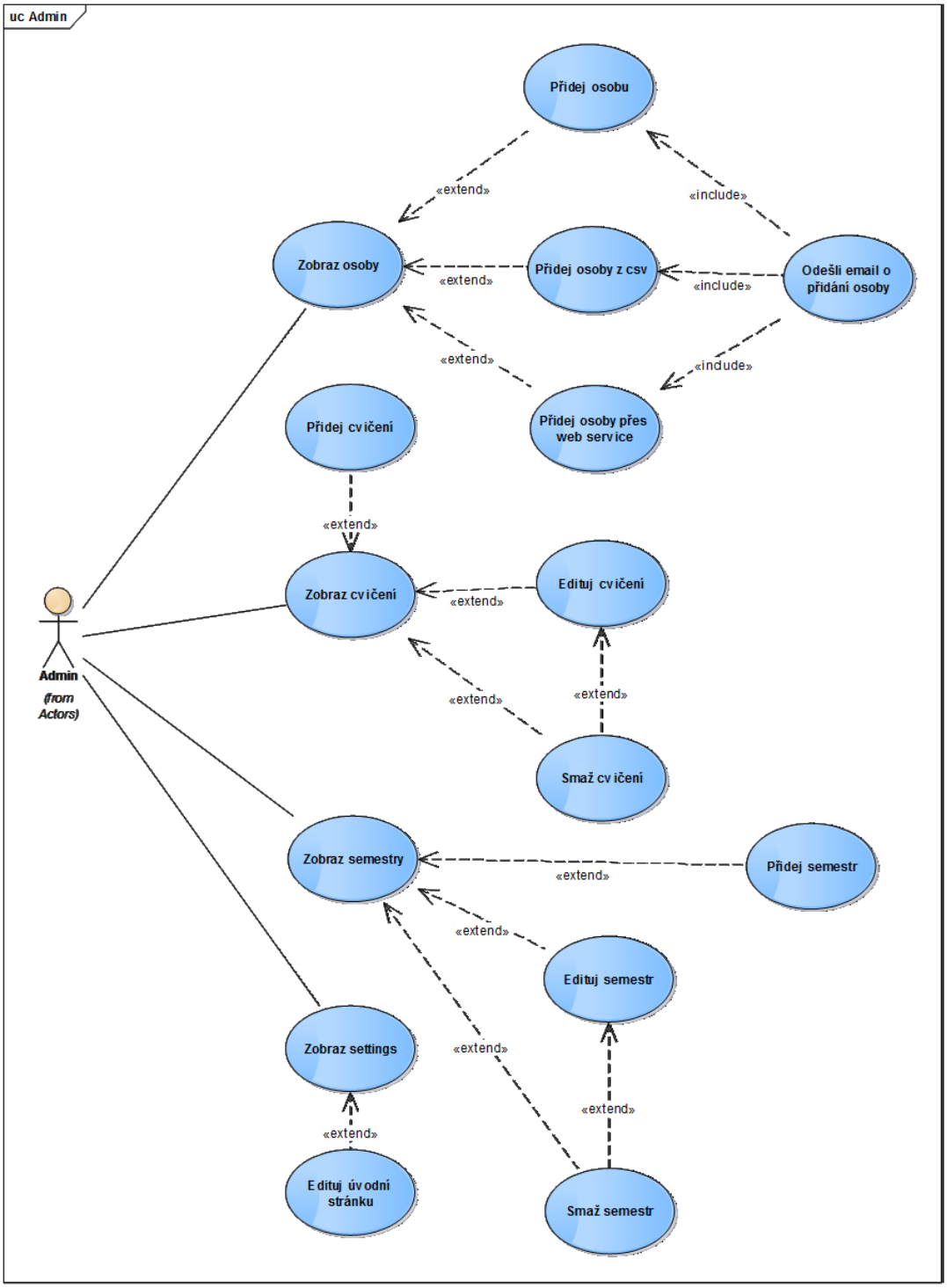
Učitel je osoba, která má v systému na starosti cvičení v rámci semestru (tato cvičení pak navštěvují Studenti). Vytváří týmy, které spravuje. Proto má tato osoba svou vlastní roli a tou je role Teacher (Učitel).



Obrázek 3.6:*Use Case -Teacher(Učitel)*

3.3.5 **Administrátor**

Jako Administrátora si můžeme představit správce systému nebo garanta předmětu. Administrátor dědí práva z Učitele, plus má opět některá práva navíc.



Obrázek 3.7: Use Case - Administrátor

3.4 Tým

Tým je soubor určitého počtu studentů, kteří společně usilovně pracují na splnění daného úkolu (projektu) pod vedením učitele. Tým je reprezentován leaderem, který komunikuje s učitelem a předává informace od učitele dál směrem ke členům týmů a naopak. Svého vedoucího týmu si volí studenti na základě konzultace s učitelem. Učitel přiřazuje studentům nebo celému týmu úkoly, které jednotliví studenti řeší, a jsou za ně hodnoceni. Vedoucí týmu má možnost přiřazovat úkoly jednotlivým členům týmu a hodnotit je.

*Jak sem již zmínil na začátku kapitoly, pro naše potřeby budu popisovat modelovaný projekt. Do našeho systému přidá učitel imaginární tým „alpha“. Ten je zařazen pod určité cvičení v daném semestru. Do týmu učitel přiřadí „studenty 1 a 2“ a vedoucího týmu „Team leader“(*v reálném životě *taky student,* ale *ne již pro systém. V systému je* již veden *v roli vedoucího týmu)*.

3.5 Projekt

Projektem je myšlen úkol, který dostane tým, jenž má během semestru udělat. Projekt jako takový je rozdělený na části, které se musí splnit.

Bude se tedy jednat o projekt „Setkání s potencionálním zákazníkem za účelem vytvoření informačního systému“. Pro tento projekt buď učitel použil hotovou šablonu, kterou si již předdefinoval nebo vytvořil nový projekt speciálně pro daný tým.

3.6 Část projektu

Každá část projektu se skládá z jednotlivých úkolů, které plní studenti. Jakmile jsou všechny úkoly hotovy, je i část projektu hotova.

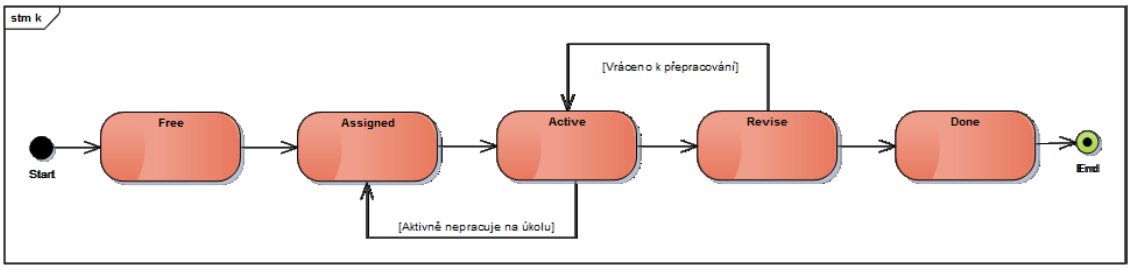
*Jednotlivé části úkol*u *v našem projektu tedy budou:*

- *Příprava na jednání*
- *Sběr požadavků zákazníka*
- *Vyhodnocení*
- *Zápis z jednání*
- *Prezentace pro zákazníka*

Přičemž žádná část projektu neobsahuje zatím žádné úkoly.

3.7 Úkol

Jednotlivé úkoly jsou vypisovány učitelem nebo vedoucím týmu. Pokud je úkol vypsán učitelem, tak spadá do třídy hodnocení učitele. Pokud je úkol vypsán vedoucím týmu, tak hodnocení spadá do třídy hodnocení týmu.



Obrázek 3.*8*: *Flow - Stav úkolu*

Úkol má cekem 5 stavů. Free, Assigned, Active, Revise a Done:

Free znamená, že na úkol se může přihlásit libovolný člen týmu a začít na něm pracovat. **Assigned** úkol je již přiřazen jednotlivému studentovi učitelem nebo leaderem. Student, ale zatím na úkolu nepracuje. Úkol se stává **Active** ve chvíli, kdy student začne na úkolu pracovat. Pokud je úkol ve stavu **Revise**, bude učitelem či leaderem úkol zkontrolován. Pokud úkol splňuje požadavky je ohodnocen a nastaven do stavu hotový. Pokud požadavky nesplňuje, je vrácen studentovi a nastaven do stavu zadán. Úkol ve stavu **Done** je patřičně ohodnocen a již se na něm nepracuje.

3.7.1 Zadání úkolu

Při zadání úkolu bude třeba jej zařadit pod příslušnou část projektu. Napsat název úkolu, maximální počet bodů, který může student získat, hodinovou dotaci a deadline (tedy termín, do kdy musí být úkol hotový).

Volitelná bude specifikace zadání, připojení souboru, komentáře.

Vezměme si tedy část projektu „1. Příprava na jednání“ a přidejme jednotlivé úkoly. Jako první úkol zadá učitel studentovi 1, aby zjistil informace o klientovi, se kterým se má setkat. Úkol bude hodnotit učitel, má tedy třídu hodnocení učitel.

Aktuální stav úkolu:

- Část projektu: 1. Příprava na jednání
- Název úkolu: Informace o klientovi
- Specifikace: Zjisti, co nejvíce informací (Finanční úřad, současní klienti, atd.)
- Bodové hodnocení: 5 bodů
- Hodinová dotace: 2 hodiny
- Datum startu: 8. 2. 2015
- Datum odevzdání: 15. 2. 2015
- Stav: Zadán studentovi 1

Vedoucímu týmu učitel podobně *zadá* přípravu *podkladů pro případný budovaný informační systém. S bodovým ohodnocením 10 a hodinovou dotací 3 hodiny. Opět je tento úkol* ve *třídě hodnocení učitel.*

Poslední úkol přidá vedoucí týmu studentovi 2. J*edná se* o *zarezervování místnosti na setkání s klientem. Tento úkol opět specifikuje, přidá bodové hodnocení 5, hodinovou dotaci 2 hodiny a datum odevzdání. Úkol náleží do třídy hodnocení vedoucí týmu.*

Pro rekapitulaci 1. Část projektu se tedy skládá z 0/2 hotových úkolů, 0/15 bodů, 0/5 hodin práce a 0 bodů osobního hodnocení části projektu. V tomto případě se jedná a třídu hodnocení učitel, jelikož má větší prioritu. Pokud bychom sloučili třídy hodnocení dohromady 1. č*ást projektu by se skládala u 0/3 hotových úkolů, 0/20 bodů, 0/7 hodin práce a 0 bodů osobního hodnocení části projektu. Toto sloučení vznikne tím, že učitel povýší úkol „Zarezervování místnosti" do kategorie hodnocení učitel. Tento úkol shledal učitel natolik důležitý/nezbytný/vhodný pro začlenění do třídy hodnocení učitel, která má vliv na hodnocení projektu.*

3.7.2 **Odevzdání úkolu**

Při odevzdání úkolu student či vedoucí týmu nastaví úkol do stavu ke kontrole a zadají čas (hodiny), který nad prací strávili. Volitelně budou moct připsat poznámku nebo připojit soubor.

Následně učitel nebo vedoucí týmu, podle toho, do které třídy hodnocení úkol spadá, úkol zkontrolují. Můžou úkol studentovi vrátit, tím se začíná počítat kvalita práce související s počtem vrácení, nebo úkol označit jako hotový a ohodnotit vykonanou práci.

Pokud by chtěl student v našem modelovaném systému odevzdat úkol, tak přiloží popisek či soubor k úloze, napíše čas (počet hodin), který strávil na úkolu (student nevidí kolik je maximální hodinová dotace) a přepne stav úkolu „ke kontrole" a potvrdí. V tuto chvíli indikátory učiteli oznamují*, že student 1 splnil úkol „Informace o klientovi" a může jej zkontrolovat. Následné hodnocení probíhá zkontrolováním popisku či dokumentu. V tuto chvíli mohou nastat dvě varianty:*

- *První varianta je kladná. Učitel popisek či dokument zkontroluje* a s*hledá jej dostatečným, bodově ohodnotí a přepne do stavu hotovo. Pokud bude* vypracování opravdu *významné či nápadité, je možné k úkolu (tedy studentovi) přidat body do osobního hodnocení. Tyto body nebudou mít vliv na celkový počet bodů týmu, ale pouze* na *celkový počet bodů studenta*
- *Druhá varianta je* záporná. U*čitel* po kontrole úkol *studentovi* vrátí*, jelikož neobsahuje dostatečné informace. Tím by se stav úkolu přepnul znovu na stav zadán a přičetl by se inkrement počtu vrácení o 1. Student by dále pracoval na projektu, ale při další kontrole již učitel vidí, že práce byla* již *jednou* zkontrolována *a* vypracování tedy *zabralo více času. Tyto „chyby" (kvalita práce) díky indikátorům učitel při druhém hodnocení vidí, a může* tak *studenta (úkol) penalizovat v osobním hodnocení.*

3.8 Třídy hodnocení úkolů

Máme dvě kategorie hodnocení úkolu. Pokud je úkol vypsaný učitelem, tak splňuje pevně dané požadavky na jednotlivé části projektu a tudíž je prioritně důležitý pro celkové hodnocení a patří do třídy hodnocení učitelem.

Úkoly vypisované vedoucím týmu, mají nižší prioritu hodnocení. Pro představu se jedná o úkoly, které jsou potřebné pro fungování potřeb týmu. Tyto úkoly mohou vzniknout i přirozenou cestou na základě diskuze mezi učitelem a vedoucím týmu.

Pokud budou úkoly vypsané vedoucím týmu a významné v rámci projektu, mohou být učitelem povýšeny do třídy hodnocení učitelem.

3.9 Hodnocení

3.9.1 Projekt

Základním elementem pro hodnocení projektu jsou úkoly. Ze splněných a ohodnocených úkolů se poté skládá hodnocení jednotlivých částí projektu a následně z celého projektu.

Bodový systém

Abychom dokázali jednotlivé úkoly patřičně ohodnotit, musí existovat bodový systém. Úkol tedy bude mít při jeho vytvoření stanovenou maximální hranici. Ve chvíli kdy je úkol hotov, je podle toho, jak bylo splněno zadání, ohodnocen.

Hodinový systém

Úkol má přidělen počet hodin, které by na úkolu měl student strávit. Tento hodinový příděl bude stanoven učitelem nebo vedoucím týmu. Čas, který student stráví na úkolu, bude v celkovém součtu představovat čas, který strávil na projektu a je jedním z faktorů při ovlivňování kvality práce a hlavně jednotlivce.

Počet vrácení

Dalším z faktorů, které ovlivňují kvalitu úkolu (práce) je počet vrácení úkolu. Pokud student odevzdá úkol, který nesplňuje zadání či představy zadavatele, musí být vrácen učitelem nebo vedoucím k předělání. Tato informace je důležitým milníkem hodnocení kvality.

Osobní hodnocení

Pro celkové hodnocení je velice důležité i osobní hodnocení. Úkoly jsou přiřazené jednotlivým studentům a lze k úkolům (tedy jednotlivcům), připsat bonusové body nebo penalizace. Toto hodnocení se promítne do bodového systému. Osobní hodnocení je velice sugestivní a může zastávat velice rozsáhlé spektrum příčin a následků, které budou sloužit jako prostředek učitele.

Deadline

U každého úkolu je třeba hlídat do kdy, se má odevzdat. Pokud je odevzdán po termínu, je tuto informaci třeba vzít na zřetel a přičíst ji do celkové kvality práce

3.9.2 Tým

Tým je hodnocen za celkovou práci na projektu. Jedná se tedy o součet bodů za jednotlivé úkoly v rámci jednotlivých částí projektu. Za části na projektu je zodpovědný vedoucí týmu. Je tedy možné penalizovat či přidávat bonusové body v rámci bodového systému vedoucímu týmu za jednotlivé části projektu.

3.9.3 Jednotlivce

Hodnocení jednotlivce se skládá ze součtu bodů, které byly studentovy přiděleny učitelem za hotové úkoly do bodového systému. Práce na jednom úkolu je ohodnocena na základě součtu osobního hodnocení a bodů, které získal za splnění úkolu. Tyto hodnoty jsou vloženy učitelem na základě odvedené práce a díky ukazatelům kvality práce (hodinový systém, počet vrácení, deadline).

3.10 Vizualizace hodnocení

Popíši zde pohled učitele na úkol jako takový a seznam úkolů. Nebudu zde popisovat, co uvidí vedoucí týmu či student, jelikož se jedná o stejnou vizualizaci, při které neuvidí určité elementy (například hodinový příděl).

Jakmile si učitel zvolí seznam úkolů, uvidí u každého úkolu indikátory, které budou naznačovat, že je něco v nepořádku. Tedy například úkol má již po termínu nebo vysoký počet vrácení. Jakmile budou všechny hodnoty potřebné pro výpočet hodnocení zadány, systém zobrazí navrhnuté hodnocení. Toto zobrazení bude jak u seznamu úkolů, tak i u jednotlivých úkolů.

Projekt

Jakmile si učitel zobrazí vybraný projektu, uvidí jednotlivé části projektu a v jakém jsou stavu. To znamená kolik úkolů z jedné části je hotovo, kolik hodin se na něm strávilo, hodinový příděl, kolik bodů je již získáno, bodový příděl a osobní hodnocení části projektu.

Po „otevření“ jednotlivých částí projektu se objeví úkoly, které část projektu obsahuje a obdobně jako u jednotlivých částí projektu uvidí učitel v jakém stavu je úkol, kolik hodin se na úkolu dělalo, hodinový příděl, počet získaných bodů, bodový příděl a počet vrácení.

Tým

Pro srovnání jednotlivých členů týmu mezi sebou je třeba vizuálně vidět jejich výkony vedle sebe. Jednotlivé položky, jako je počet úkolů, na kterých kdo pracoval, počet hodin, které celkově strávili na úkolech, kolikrát překročil student deadline, bodové hodnocení a osobní hodnocení, budou sloužit jako prostředek konfrontace odvedené práce mezi členy týmu.

Jednotlivec

Při pohledu na jednotlivce je třeba vidět, na kterých úkolech student pracoval či pracuje. Tedy, které úkoly mu byly přiděleny. Jejich hodnoceni a kvalita (hodinový systém, počet vrácení, deadline).

3.11 Nástěnka

Nástěnka slouží jako prostředek ke sledování činností, které se dějí v týmu, které se děly a které se budou dít. Nástěnka je místem, kde student hned vidí, kdo na čem pracuje a co se chystá. Tedy úkoly a události.

Kalendář

Kalendář slouží jako orientační prvek, díky kterému bude možné sledovat události, které se dějí v rámci semestru, cvičení či v týmu. Student hned vidí, kdy probíhají týmové schůzky nebo případný test v rámci cvičení.

Událost

Událostí se myslí akce, která probíhá v určitou dobu. Tedy například schůzka, návštěva muzea, termín odevzdání všech dokumentů. Události jsou studentovi přístupné skrze nástěnku, aby byl hned informován o nadcházejících událostech.

3.12 Šablony projektů

Učitel má možnost vytvořit šablony, ve kterých bude definovat projekt, jeho části a úkoly a používat je pro různé projekty či další týmy v rámci jiných semestrů.

3.13 Dokumenty

Systém bude ukládat dokumenty ve stylu repositáře, kdy bude vždy aktuální verze přístupná. Starší verze budou archivovány.

Struktura

Dokumenty budou uloženy na serveru a je zde potřeba vytvořit strukturu, která bude přehledně reprezentovat týmy a úkoly. Semestr obsahuje týmy. Typy souborů v týmu mohou být celkem tři. Soubor spjatý s úkolem, soubor který sdílejí členové týmu mezi sebou a soubory, které odevzdává leader učiteli.

Týmové úložiště

Pro interní potřeby týmu bude k dispozici úložiště, kde budou studenti moci sdílet své dokumenty. Pokud například budou chtít členové týmu soubor jakéhokoliv typu sdílet mezi sebou, mohou tak učinit skrze toto úložiště.

Leader úložiště

Pouze leader má k dispozici toto úložiště. Jedná se o úložiště, do kterého leader ukládá soubory, které je třeba odevzdat učiteli. Jsou to soubory s vyšší důležitostí, které je třeba zkontrolovat za celý tým nebo větší část projektu. Učitel si pak může, ať už přes webové rozhraní, nebo přes ftp, tyto dokumenty stáhnout.

3.14 Diskuze

Tým bude mít k dispozici diskuzi, kde si budou moci členové sdělovat své poznámky, nápady a připomínky týkající se jejich týmové spolupráce. Členové týmu budou moci sdílet své nápady skrze aplikaci, která uchová i historii probíhající diskuze.

4 Vymezení metodiky vyhodnocování v závislosti na realizovaném systému

Smyslem systému je pomoci studentům v jejich práci, komunikaci a sebeurčení. Nesmíme ale zapomínat na učitele, kteří tráví hodiny a dny nad tím, aby připravovali podklady pro svou výuku, opravovali práce, a kteří obětují svůj volný čas při pomoci studentům. Budeme se tedy snažit učitelům pomoci nést jejich břímě. Nebylo by krásné mít systém a v něm implementovaný algoritmus, který vyhodnotí práci studentů za ně? Buďme upřímní, vždy bude třeba lidského dohledu, abychom ohodnotili práci studentů. Nemůžeme s nimi zacházet jako s dalším číslem na listu. Můžeme ale využít autonomního vyhodnocování k návrhům známek, hodnocení a kontrole. V této kapitole si tedy probereme konkrétní řešení problematiky autonomního vyhodnocování úkolu. Abychom se ale k tomuto cíli dostali, musíme nejprve získat data, ze kterých budeme vycházet. Takováto data je třeba nejprve předzpracovat a následně filtrovat. Dostaneme tak vstupy, ze kterých můžeme dále vycházet a normalizovat je. Ve chvíli, kdy máme normalizovaná data, je pak otázkou, jaké metody a postupy zvolit, abychom dosáhli cíle.

4.1 Předzpracování dat

Předtím, než začneme pracovat se samotnými daty, musíme si je upravit tak, abychom byli schopni s nimi pracovat. Musíme se zbavit nežádoucích dat, a to do takové míry, aby bylo možné zbývající data vhodně zpracovávat.

Obecně se takovému procesu říká předzpracování dat [25]. Příkladem nevhodných záznamů jsou například neúplné výsledky. Tyto hodnoty pak dále nahrazujeme, přepočítáváme nebo případně přepisujeme do takové podoby, aby bylo možné s nimi pracovat.

4.2 Filtrace dat

Jak jsem zmínil výše, ne všechna data jsou vhodná pro zpracování [26]. Pokud tedy vezmeme úkol jako takový, musíme se podívat, zdali má vyplněny všechny požadované hodnoty, které jsou potřeba pro jeho vyhodnocení. V opačném případě tato data nemá smysl hodnotit. Pokud rychlost zpracování úkolu, počet bodů nebo počet vrácení nebude mít vyplněnou hodnotu, nemá pak takováto hodnota pro nás žádnou vypovídající hodnotu a nemusíme jí zahrnovat do konečného zpracování.

4.3 Co jednotlivé hodnoty představují

Vstupy

- Odhad – čas potřebný ke splnění úkolu
- Spáleno – čas, který osoba potřebovala ke splnění úkolu
- Osobní body – osobní ohodnocení zadavatele

- Max bodů – maximální počet bodů
- Sum bodů – počet bodů, kterou dostala osoba za úkol
- Vráceno – počet vrácení

Vypočítané vstupy

- Potřebný čas – čas potřebný ke splnění úkolu
- RychlostN – normalizovaná rychlost, kterou student splnil daný úkol
- BodyN – normalizovaný celkový počet bodů, který student získal
- VrácenoN – normalizovaný počet vrácení

Trénování učitelem

- Výsledek – známka přiřazená učitelem
- Výsledek% – výsledek ohodnocen učitelem v procentech

Výstup

- VýsledekN – normovaný výstup Výsledek%, trénovaného učitelem
- Návrh – navrhnutý výsledek

Data

Odhad	Spáleno	Osobní body	Max bodů	Sum bodů	Vráceno	Potřebný čas	RychlostN	BodyN	VrácenoN	Výsledek	Výskledek%	VýsledekN	Návrh
50	25	0	10	0	0	0,5	0,997527377	0	0,997527377	5	0	0	0
50	48	0	10	10	0	0,96	0,962312109	1	0,997527377	1	100	1	0,959932674
50	66	1	10	9	0	1,32	0,746493983	1	0,997527377	2	80	0,8	0,744648185
6	10	0	10	0	0	1,666666667	0,268941421	0	0,997527377	5	0	0	0
1	1	0	10	1	1	1	0,952574127	0,1	0,98201379	5	10	0,1	0,093544093
2	1	0	10	3	0	0,5	0,997527377	0,3	0,997527377	4	25	0,25	0,29851826
10	8	2	10	8	0	0,8	0,985225968	1	0,997527377	1	100	1	0,982789876
1	3	0	10	4	0	3	0,000123395	0,4	0,997527377	5	1	0,01	4,92358E-05
1	2	0	10	0	0	2	0,047425873	0	0,997527377	5	0	0	0

Tabulka 4.1: Výtah z testovacích dat

4.4 Automatické doplnění hodnot

Pokud vezmeme hodnoty, které používáme k hodnocení studenta pro daný úkol (tedy rychlost, celkový počet bodů a počet vrácení), je zřejmé, když student neodevzdá úkol v daném čase (nebo v nejzazším možném termínu), tak bude jeho hodnocení nahrazeno. Například pro záznamy bodového hodnocení to znamená, že ho nahradíme nulou. Takto nahrazená hodnota zajistí integritu bodového hodnocení pro studenta, který bude nakonec zařazen do finálního zpracování. Pokud student úkol neodevzdá, nemá smysl řešit počet odevzdání, výsledek je tedy srovnatelný s případem, kdy úkol neodevzdá. Pokud celkový počet bodů, který student získá, je roven nule, tak výsledek je zřejmý.

4.5 Normalizování hodnot

Konečné hodnocení studentova úkolu je sice na vyučujícím, avšak mým úkolem bylo umožnit vyučujícímu využít autonomního systému hodnocení úkolů. V konečném důsledku to

pak znamená, že pokud učitel zadá hodnocení úkolů z jednotlivých hledisek, využije můj systém znalostí z předchozích hodnocení a výpočtem pak navrhne hodnocení vlastní na základě nějaké běžně užívané metody, nebo na základě vlastního uvážení.

Běžnou praxí pro předání hodnot do zpracující metody je její převedení do určité normované podoby [25] tak, aby všechny vstupní hodnoty měly přibližně stejný rozsah a jedna hodnota výrazně nepřevažovala nad hodnotami ostatních vstupů, což by mohlo negativně ovlivnit celkový výsledek.

Například převádíme-li vstup booleovského typu (pravdivostní hodnotu), bude normovaná hodnota pro nepravdu rovna nule a pro pravdu rovna jedné. Takovým speciálním případem pravdivostní hodnoty je fuzzy logická [27] hodnota, která udává takzvanou míru pravdivosti (např. Sklenice je napůl plná odpovídá pravdivostní hodnotě 0,5 pro "plnost sklenice"). V našem kontextu to pak může znamenat, že pokud student splnil 50 % úkolu, je jeho bodové hodnocení rovno 0,5, což jistě popisuje skutečnost lépe, než pravdivostní hodnota splnil/nesplnil (0/1). Vzhledem ke zmíněným poznatkům by se pak dalo zjednodušeně říci, že se snažíme přepočítat proměnné hodnoty vstupu do rozsahu mezi 0 a 1.

4.5.1 **Rychlost práce, bodové ohodnocení a počet vrácení k přepracování**

V mém systému hodnocení využívám tři hlavní indikátory (ukazatele), a to rychlost dokončení úkolu (dále jen rychlost), bodové hodnocení a počet navrácení úkolu k přepracování (dále jen vracení). Normalizace bodového hodnocení jsem uvedl v předchozím odstavci. Problém však nastává v případě rychlosti a vracení. Nejjednodušším řešením normalizace pro tyto vstupní hodnoty by bylo použití běžných pravdivostních hodnot pro rychlost "odevzdal včas" (ano/ne) a pro vracení „překročil počet vracení" (ano/ne). Já jsem však chtěl více sofistikovat výsledky, kterých z těchto hledisek studenti dosahovali, a využil jsem pro normalizaci takzvanou logistickou funkci.

4.5.2 **Logistická funkce a její využití**

Logistická funkce nebo také logistická křivka, dokáže vyjádřit proměnlivost hodnot například v čase. Vzorec pro výpočet funkce je[24]:

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}} \tag{4.1}$$

Kde:

L – je maximální hodnota (konstanta)

e – je Eulerovo číslo (přirozený logaritmus)

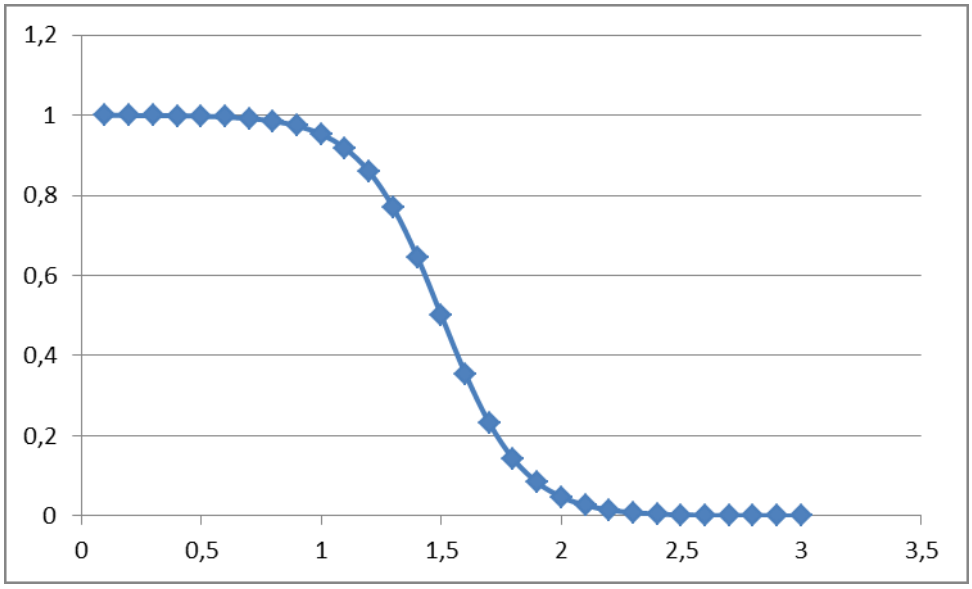
k – zakřivení křivky (konstanta)

x – proměnná v rozsahu reálných čísel

x0 – hodnota x-tého prvku (konstanta)

Nyní tedy využijeme logistické křivky při hodnocení rychlosti a počtu vrácení.

Na obrázku vidíme závislost potřebného času na hodnocení. Osu x představuje doba, kterou student potřeboval na splnění zadaného úkolu. Pokud tedy čas potřebný na splnění úkolu byl 10 hodin a student zadaný úkol přesně splnil za tuto dobu, tak na ose x představuje tato hodnota bod 1. Odpovídající ohodnocení na ose y se pak blíží k jedné, ale jedná se spíše *0,95*.



Obrázek 4.1: *Závislost času na hodnocení*

Požadovaného obrázku jsme docílili, jak již bylo zmíněno, pomocí logistické funkce, a to tak, že:

$$f(x) = \frac{1}{1+e^{-6(1,5-1)}} \tag{4.2}$$

Kde jednotlivé hodnoty představují:

- L – 1, tedy normalizovaná maximální hodnota, které může student dosáhnout
- k – -6, díky které jsem dosáhl požadovaného zakřivení
- x – 1,5, která mi představuje jakousi střední hodnotu, kde již překlápím hodnocení studenta k nižším hodnotám (horší známce v důsledku)
- x0- 1, tedy potřebný čas (10 hodin z 10 hodin)

Jako výsledek logistické funkce dostanu hodnotu *0,952574127*. Tato hodnota mi pak slouží jako indikátor rychlosti, kterou student potřeboval ke splnění úkolu, a tedy jako jeden ze vstupních parametrů hodnocení.

Obdobně pak pokračuji se spravováním hodnoty počtu vrácení, kde pak opět pomocí logistické funkce dostanu další vstup do parametrů hodnocení. Jako další vstupní parametr

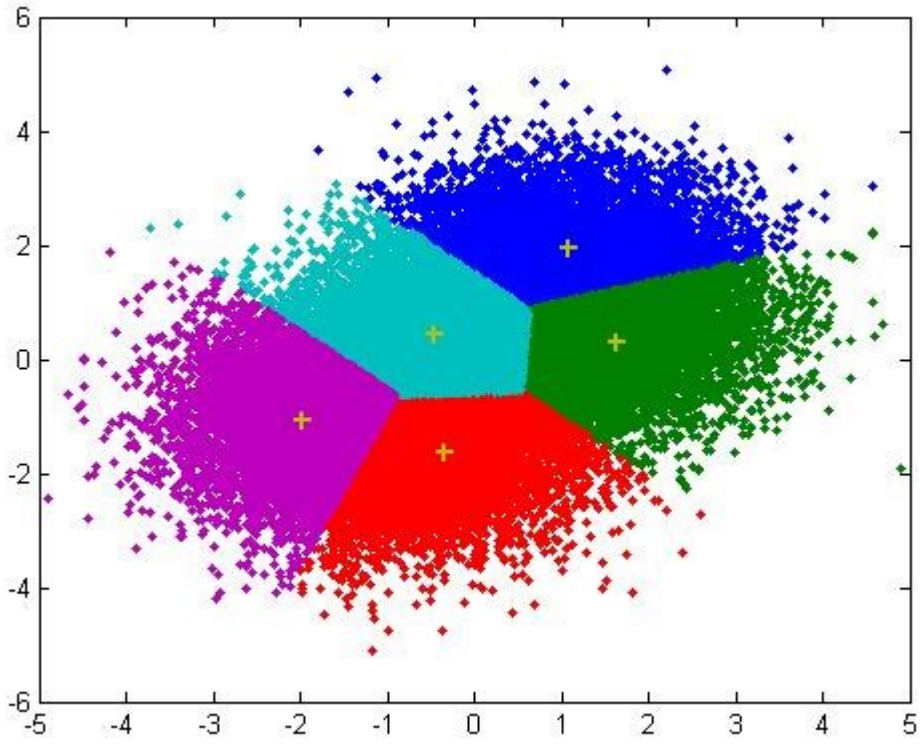
slouží parametr celkový počet bodů, který dostaneme součtem bodů a jejich následného vydělení maximálním počtem bodů.

4.6 Algoritmus vyhodnocování

V předchozí kapitole v Tabulce 4.1 je uveden také sloupec označený jako Návrh. Jedná se o jednoduchý výpočet pro navrhnutí známky, který vychází z aktuálně ukončeného úkolu (tedy úkolu ve stavu Done). Za předpokladu, že všechny vstupní hodnoty budou odpovídat podmínkám zpracování (viz. Předchozí kapitola), můžeme práci studenta jednoduše vyhodnotit součinem normalizovaných vstupních hodnot a navrhnout známku učiteli. Tedy $rychlost \cdot body \cdot vráceno$. Takováto metoda vyhodnocování je ale velice nepřesná. Z tohoto důvodu se podíváme po jiných metodách, které by nám pomocí učících postupů dokázaly zpřesnit a vyhodnotit práci/úkol/známku přesněji. Jednou z nich je metoda hledání nejbližšího souseda.

4.6.1 Hledání nejbližšího souseda

Metoda hledání nejbližšího souseda je strojový algoritmus, který se používá pro rozpoznávání vzorů nebo také společných vlastností v rámci určitého prostoru. Jednotlivé prvky vstupů mohou být rozptýleny v N-dimenzích, které ve skutečnosti představují vektory v tomto prostoru [21]. V rámci učení se pak tyto vektory zpracovávají do jednotlivých množin (clusterů), kde pak střední hodnotu jednotlivé množiny představuje centroid (nebo také reprezentant množiny) [22]. V následující klasifikaci se pak algoritmus, například pomocí Euklidovy vzdálenosti snaží určit, do které množiny zadaný vstup patří.



Obrázek *4.2*: *Množiny (Clustery) metody k-means* [33]

Z diagramu jsou vidět jednotlivé množiny (clustery), které například odpovídají množinám jedniček, dvojek, atd. V těchto shlucích, pak existují jednotliví reprezentanti množin.

4.6.2 Trénování s učitelem

K tomu, abychom měli data, z kterých se můžeme učit (trénovat), nám poslouží data, která jsem zmínil v předešlé kapitole, a která jsou již ohodnocená učitelem, a to jak celkovou známkou, tak i bodově (procentuálně). Tyto hodnoty najdeme ve sloupečcích jako výsledek a procentuální výsledek. Čím více jich tedy budeme mít, tím přesnějších výsledků můžeme docílit.

4.6.3 Euklidovská vzdálenost

Jak jsem již zmínil, metoda hledání nejbližšího souseda může využít pro hledání representanta množiny Euklidův algoritmus [8]. Euklidův algoritmus složí k tomu, aby nám dokázal najít vzdálenost mezi dvěma body v trojrozměrném euklidovském prostoru. Kde p a q představují jednotlivé body v prostoru.

$$(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \tag{4.3}$$

Pokud převedeme algoritmus na výpočet našeho konkrétního úkolu, přičemž vstup by byl reprezentantem dané množiny a aktuálně vyhodnocovaný vstup (tedy takový, který se snažím přiřadit do správné množiny), algoritmus by vypadal následovně:

$$d(reprezentant, ukol) = \sqrt{\begin{matrix}(rychlost\ reprezentanta - rychlost\ úkolu)^2 + \\ (body\ reprezentanta - body\ úkolu)^2 + \\ (p.vrácení\ reprezentanta - p.vrácení\ úkolu)^2\end{matrix}} \quad (4.4)$$

Stejný algoritmus použijeme i na nalezení procentuálního výsledku hodnocení. Pokud tedy máme reprezentanta dané množiny, máme i jeho minimální a maximální procentuální hodnotu výsledku. Díky tomu dokážeme najít vzdálenost mezi aktuálně vyhodnocovaným vstupem a jeho vzdálenost mezi minimálním a maximálním výsledkem.

4.7 Porovnání výstupů

Vzali jsme tedy vstupní data (s hodnocením úkolů učitelem) a použili jsme je k trénování našeho klasifikátoru (algoritmu pro vyhodnocování). V tomto momentu jsme získali reprezentanty množiny a můžeme spustit testovací data nad algoritmem vyhodnocování. Dostaneme tak navrhnuté výsledky, které vidíme v tabulce 4.2. Vezmeme-li hodnocení učitele a výsledky vyhodnocovacího algoritmu, u kterých následně uděláme absolutní rozdíl a kvadratický rozdíl, dostaneme hodnoty, které nám řeknou, zda-li se výsledky liší. Výsledky máme jak pro hodnoty známkovací, tak pro hodnoty bodové. Nakonec si uděláme průměry výsledných rozdílů, které jsou zobrazeny v tabulce 4.3. První hodnota SUM reprezentuje součet rozdílů hodnot známek a bodového rozdělení. MSE je střední kvadratická odchylka a ME průměr výsledků.

Pokud se podíváme na průměrné hodnoty výsledků, vychází nám, že při celkovém počtu 200 natrénovaných hodnot, se nám navržená známka průměrně rozchází o *0,927669* stupně. Pokud bychom si nechali algoritmem navrhnout bodové hodnocení, máme zde rozdíl *15,9158837* bodů (procent).

Vzhledem k počtu dat použitých na natrénování reprezentantů množin a počítání Euklidovské vzdálenosti si myslím, že jsme se dobrali k relevantním výsledkům.

Pokud bychom chtěli ještě více zlepšit přesnost vyhodnocování, pak toho můžeme docílit minimálně dvěma způsoby.

Zvětšením počtu vstupních dat potřebných k trénování. Ideálně pokud by jeden semestr výuky, pro který je tato aplikace stavěna, již proběhl, a my bychom tak měli k dispozici potřebné množství dat (samozřejmě čím více, tím lépe).

Další možností je zaměnění výpočtu Euklidovy vzdálenosti za například Manhattanskou metriku nebo Hammingtonovu vzdálenost, tedy změna výpočtu vzdálenosti.

Vyhodnoceno algoritmem						Vyhodnoceno učitelem						Rozdíl hodnot			Kvydratický rozdíl	
RychlostN	BodyN	VrácenoN	Hodnocení	Hodnocení %		RychlostN	BodyN	VrácenoN	Hodnocení	Hodnocení %		Rozdíl	Rozdíl%		Rozdíl	Rozdíl %
0,9975274	0	0,9975274	5	0		0,997527377	0	0,997527377	5	0		0	0		0	0
0,9623121	1	0,9975274	1	100		0,962312109	1	0,997527377	1	100		0	0		0	0
0,746494	1	0,9975274	2	87,84421		0,746493983	1	0,997527377	2	80		0	7,84421		0	61,53163
0,2689414	0	0,9975274	5	12,15332		0,268941421	0	0,997527377	5	0		0	12,15332		0	147,7032
0,9525741	0,1	0,9820138	5	2,293892		0,952574127	0,1	0,98201379	5	10		0	7,706108		0	59,3841
0,9975274	0,3	0,9975274	4	5,678362		0,997527377	0,3	0,997527377	4	25		0	19,321638		0	373,3257
0,985226	1	0,9975274	1	98,21139		0,985225968	1	0,997527377	1	100		0	1,78861		0	3,199126
0,000123395	0,4	0,9975274	3	16,22862		0,000123395	0,4	0,997527377	5	1		2	15,22862		4	231,9109
0,04742587	0	0,9975274	5	14,11498		0,047425873	0	0,997527377	5	0		0	14,11498		0	199,2327
0,9525741	0	0,9975274	5	0,9328141		0,952574127	0	0,997527377	5	0		0	0,9328141		0	0,870142

Tabulka 4.2: Navrhnuté výsledky

SUM	191	56236,0086
MSE	0,86036	253,3153541
ME	0,927556	15,9158837

Obrázek 4.3: Průměrné hodnoty výsledků

5 Realizace

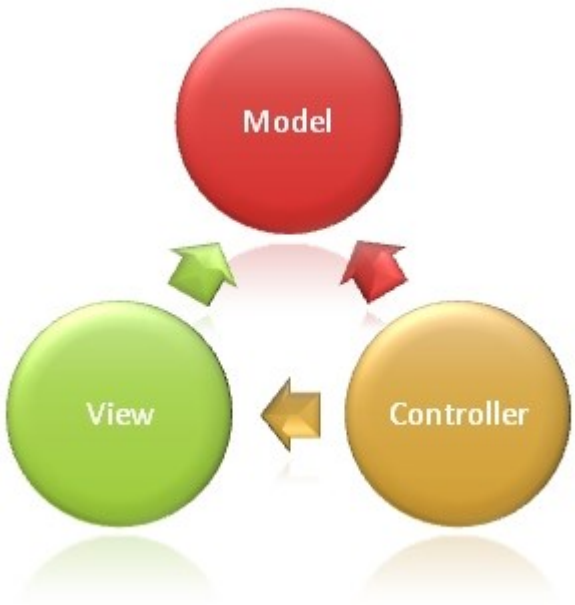
V této části práce se zmíním nejen o použitých technologiích, praktikách, ale i části zajímavé především z implementačního hlediska. Shrnu navrhnutou architekturu systému a ukáži vizualizaci aplikace. Pro lepší pochopení uvádím i části kódu.

5.1 Technologie

Pro realizaci systému, jak je popsáno v požadavcích jsem zvolil technologie firmy Microsoft. Tyto technologie jsou dlouholetou praxí ověřeny jako funkční a spolehlivé.

5.1.1 ASP.NET MVC

ASP.NET MVC je jeden za tří frameworků, který .NET podporuje pro tvorbu webových stránek. Zbývající dva zástupci jsou web formy a web pages. Framework využívá MVC vzor, neboli také Model-View-Controller. Z obrázku je pak vidět, jak jednotlivé prvky modely spolu komunikují.



Obrázek *5.1*: *MVC architektura volání* [34]

Model se stará o data a o případnou logiku (pokud je třeba). View reprezentuje data (například na webovém klientovi). Controller je zde k tomu, aby se staral o obsluhoval uživatele, tedy přebírá data z view a posílá je do modelu.

Díky takto rozdělené architektuře dokážeme oddělit business logiku od view. Dle mého názoru největší přínos MVC je jeho jednoduchost a znuvupoužitelnost. Ve chvíli, kdy si nadefinuji view, jsem schopný si typově zajistit, co se mi bude renderovat.

5.2 Praktiky

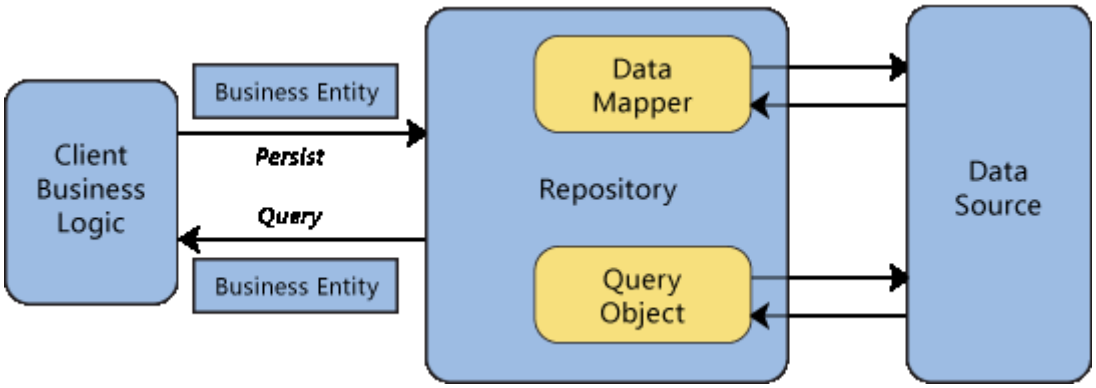
Zde bych rád poznamenal zajímavé praktiky, metodiky, návrhové - případně „architektonické“ - vzory, které jsem v realizované aplikaci použil. Nebudu zde zmiňovat základní stavební prvky aplikací a praktik obecně, jako je například použití Singleton Patternu, Factory, Fasady, aj., bez kterých se dnešní aplikace neobejdou.

5.2.1 Dependecy Injection a Autofac

Dependency injection [16] (známé jako DI), je ve své podstatě předávání závislostí do tříd (implementací). Programátor se tedy při vytváření objektů zříkává zodpovědnosti za získávání referencí, které potřebuje ke své činnosti. Tzv. se říká, že může „podhodit“ implementaci, například jednotlivým interface. O jednotlivé registrace, které interface (objekt) dostane jako implementaci, se stará dependecy container. V našem případě AutoFac (odlehčená verze Microsoft unity).

5.2.2 Repository pattern and Unity of work

Repository pattern [15] jako takový má za následek vytvoření vrstvy mezi business logikou a datovou vrstvou (v našem případě entity Framework). Na straně klienta se vytváří jednotlivé query, které se pošlou do repository, kde se nakonec poskládá do datového contextu. Tento context je na vyžádání uložen (commitnut) do databáze. Abychom ale ochránili klienta proti vytváření více repositářů (datových contextů), využíváme tzv. unity of work. Což znamená, že můžeme změnit více objektů v rámci jednoho datového contextu. Tento způsob ochrany je vhodný pro systémy, kde potřebujeme přistupovat k více objektům najednou a měnit je, aniž bychom chtěli postupně (například během jednoho http požadavku) vícekrát ukládat do databáze.



Obrázek 5.2: *Repository architektura volání* [17]

5.3 Architektura

Architekturu v aplikaci jsem zvolil třívrstvou, kde jednotlivé assembly reprezentují části systému:

- Web – prezentační vrstva
- Business – logika (obsahující strategy service)
- Data – entity Framework a repository vrstva

Zvlášť pak stojí Common assembly, která je referencovaná do všech knihoven a obsahuje společné prvky používané napříč všemi knihovnami.

5.3.1 Prezentační vrstva

View

Jak sem již zmínil, využívám nejnovější technologii ASP.MVC 5., k němuž náleží engine Razoru, a ne již zastaralý webform engine, který mvc využívalo v dobách začátků MVC2. View mám typově závislé. Jak jde vidět z kódu, k prereprezetaci dat využívám viewModely, pro které mám pak v Shared složce připravené template pro generování DOMu.

Za zmínku stojí využití T4 generátotů, které mi vytváří silnou typovou závislost mezi jménem metod controllerů a view. Tímto přístupem dokáži eliminovat chyby spojené se změnami názvů view, controleru, akcí. Aplikace je psaná v anglickém jazyce, nicméně aplikace využíva Translation resource, které se dají přepnout podle požadované kultury. Tedy například z en na cs-cz.

```
@using DP.Web
@model DP.Business.ViewModels.PersonViewModel
<h2>@Translations.PersonEdit</h2>
<div>
    <div>
        <h4>@Translations.PersonHeader</h4>
        <hr />
        @using (Html.BeginForm(MVC.Person.ActionNames.Edit, MVC.Person.Name))
        {
            @Html.AntiForgeryToken()
            <div class="form-horizontal">
                @Html.EditorForModel()
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <button type="submit"
                                class="btn btn-default">
                            @Translations.Save
                        </button>
                    </div>
                </div>
            </div>
        }
    </div>
    <div>
        @Html.ActionLink(Translations.BackToList, MVC.Person.ActionNames.Index)
    </div>
</div>
```

Controller

Jakmile potvrdíme změnu na view, odešle se na http post request na server, kde nám .NET naparsuje jednotlivé položky formuláře do viewModel. Proč ale tento základ říkám? Využívám totiž typové závislosti view na controleru, kde vím, že dostanu naplněný PersonViewModel. Využiji tohoto typu (aniž bych programově rozlišoval, o které view či controller se jedná) a zavolám obecnou metodu edit. Tato metoda přijímá jako jeden z parametrů BaseViewModel, z kterého dědí všechny viewModely a zavolám strategyservice, která rozezná, o který typ se jedná a zvolí správnou implementaci.

Jelikož využívám napříč apliakací BaseClassy, tak je vždy třeba zjišťovat, o jaký typ se jedná. K tomuto jsem si napsal extensiony, jako je například CheckBaseViewModel<PersonViewModel>(), které mi pomáhají kontrolovat typovost a zdali jsou viewModely naplněné.

```
[HttpPost]
[ValidateAntiForgeryToken]
public virtual ActionResult Edit(PersonViewModel viewModel)
{
    if (ModelState.IsValid)
    {
        var vm = _businessFactory.StrategyService.Edit(
            new BaseContext
            {
              BaseViewModel = viewModel,
              AuthenticationViewModel = ApplicationHelper.LoggedInUser
            }).CheckBaseViewModel<PersonViewModel>();

        if (vm == null)
        {
            return RedirectToErrorPage();
        }
        return RedirectToAction(MVC.Person.Index());
    }
    return View(viewModel);
}
```

5.3.2 Logika

Druhá vrstva nazvaná business (zkráceně pro business logiku), odděluje datovou vrstvu od prezentační. Proč tomu tak je? Hned z několika důvodů. Pokud například potřebuji naplnit PesonViewModel daty, které dostanu z databázové vrstvy, udělám to na jednom místě, a to v business vrstvě. Díky tomu kdykoliv budu potřebovat například změnit personu, stačí zavolat PersonService, která obsahuje potřebné metody k obsluze persony jako takové. Díky oddělení logiky od prezentační vrstvy jsem taky dosáhl využitelnosti aplikace například pro budoucího tlustého klienta, případně pro jiné prezentační rozhraní (například asp.net).

Rád bych však pokračoval v příkladu z předchozí vrstvy (chtěl bych jen poukázat na to, že StrategyService i PersonService využívají AutoFac, a tudíž i dependency injection). Nyní

jsme se dostali do StrategyService, a to přesně do metody Edit. Zde vidíme využití návrhového vzroru strategy, kde na základě typu(chování) dokáži vytvořit požadovanou instanci. Následná instance (v našem případě PersonService) obsahuje konkrétní implementaci pro obsluhu editování persony jako takové.

```
public class StrategyService : IStrategyService
{
  private readonly IBusinessFactory _businessFactory;
  private readonly IRepositoryFactory _repositoryFactory;

  public StrategyService(IBusinessFactory businessFactory,
                         IRepositoryFactory  repositoryFactory)
  {
    _businessFactory = businessFactory;
    _repositoryFactory = repositoryFactory;
  }
  …
  public IBaseViewModel Edit(IBaseContext baseContext)
  {
    var instance = GetStrategyInstance(baseContext);
    instance.Edit(baseContext);
    _repositoryFactory.Commit();
    return baseContext.BaseViewModel;
  }
}
public class PersonService : IStrategy, IPersonService
{
 private readonly IBusinessFactory _businessFactory;
 private readonly IRepositoryFactory _repositoryFactory;

 public PersonService(IBusinessFactory businessFactory,
                      IRepositoryFactory repositoryFactory)
 {
   _businessFactory = businessFactory;
   _repositoryFactory = repositoryFactory;
 }
 …

 public void Edit(IBaseContext context)
 {
   var viewModel = context.CheckBaseViewModel<IPersonViewModel>();
   var poco = _repositoryFactory.PersonRepository.Get(viewModel.Id);
     …
    _repositoryFactory.PersonRepository.Update(poco);
 }
}
```

Konkrétní implementace třídy person obsahuje již volání datové vrstvy, a to především repositáře. Pak následuje konkrétní logika, kterou zde není třeba popisovat, a uložení POCO objektu do contextu entity frameworku. Takto naplněný context již commitnem(uložíme) do databáze, kde se neděje nic jiného než SaveChanges (uložení) nad contextem Entity Frameworku.

5.3.3 Datová vrstva

Datovou vrstvu bych si dovolil rozdělit na dvě části. A to část aplikační, kterou bude tvořit repository pattern v kódu aplikace a databázová úroveň, tedy už konkrétní věci spjaté s MS SQL

Aplikační úroveň

V rámci datové vrstvy na programové úrovni využívám architektonicky vzor repository. Díky němu přistupuji k jednotlivým databázovým setům (reprezentace tabulky v aplikační vrstvě) vrstveně. Flexibilita této vrstvy je zřejmá, pakliže chceme například vyměnit entity framework za NHibernate. Na co bych chtěl v rámci implementace poukázat, je, že díky entity frameworku si generuji čisté POCO objekty, proto dokáži tyto objekty nejenom rozšiřovat, ale hlavně podědit z BasePoco, čehož využívám, když si posílám data napříč vrstvami. Pokud bych této vlastnosti nevyužil, objekty by podědily z obecné třídy entity frameworku.

Dalo by se tedy říci, že mapuji POCO objekty přímo na view modely. Což by nebylo z hlediska funkcionality nijak špatné, ale zavřel jsem si tím cestu k použití DTO k přenášení dat mezi prezenční a datovou vrstvou. V podstatě se jedná o to, že například objekt, který reprezentuje tabulku Person, tedy Person objekt, bych chtěl automaticky rozšířit o určité vlastnosti z jiných tabulek. Toho bych tedy docílil například překlapováním Person objektů na PersonDTO, která už by potřebné vlastnosti obsahovala (načetla by si je z databáze).

Já však využívám možnosti si rozšířit POCO objekty díky „partial“ implementaci na programové úrovni. Tím jsem schopný rozšířit generované objekty entity frameworkem o custom vlastnosti. Zůstala mi tedy flexibilita DTO objektů a ušetřil jsem mapování objektů na datové vrstvě.

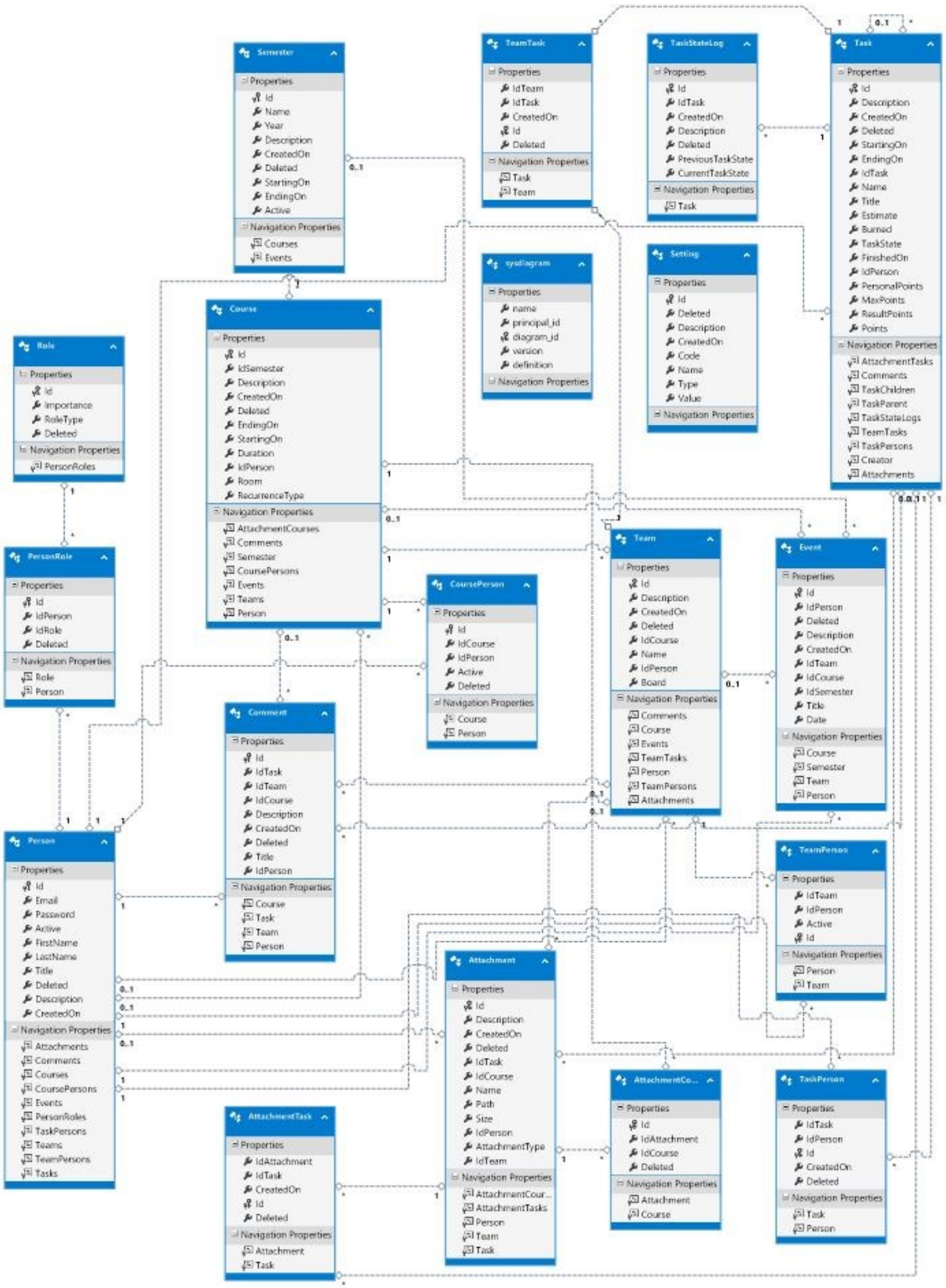
Rád bych ještě podotkl, že i na nižších úrovních stále využívám AutoFac resolvaci implementace a hlavně díky tomu dokáži řídit life-cycle databázového contextu.

Vzhledem k tomu, že nepoužívám k oddělení vrstev WCF služby a všechny vrstvy poběží na jednom serverovém stroji, tak lze napříč vrstvami posílat Queryable datový typ. Díky tomu si například z business vrstvy napíši lambda výraz pro načtení určitých dat z databáze, a až na datové vrstvě se mi výraz poskládá.

```
public abstract class BaseRepository<T> : IBaseRepository<T> where T:BasePoco
{
  #region Fields

  protected readonly IDbSet<T> _dbset;
  protected DataContext _dataContext;

  #endregion

  #region Constructor

  protected BaseRepository(IContextFactory contextFactory)
  {
    _dataContext = contextFactory.Get();
    _dbset = _dataContext.Set<T>();
  }

  #endregion
  …
  public virtual IEnumerable<T> GetMany(Expression<Func<T, bool>> where)
  {
    var result = _dbset.Where(where).ToList();
    return result.Where(x => x.Deleted == false).ToList();
  }
}
```

Databázová úroveň

Pro vytvoření databáze jsem využil metodologie zvané "Database First", což znamená navržení databáze prvně v SQL Managment studiu a následné vygenerování databázového schématu do aplikace. Zajímavá je tabulka TaskStateLog, pro kterou mám napsaný trigger, reagující na změny v sloupečku stavů (TaskState) v tabulce Task. Každá tabulka obsahuje sloupec Deleted, což je stav, který uvádí, zda-li byl záznam smazán. Tím si dokáži ošetřit stav, kdy již klient pokládá záznamy za smazané, ale zároveň dokáži uchovávat jejich historii. Takovéto mapování si pak ošetřuji jak při automatickém mapování, tak na aplikační úrovni.



Obrázek 5.3: *Databázová struktura*

5.4 Modul metodiky hodnocení

Metodika vyhodnocení je implementována jako samostatný projekt. Lze jej tedy připojit k projektu nebo přidat jako knihovnu, a následně volit požadované metody vyhodnocování. Modul obsahuje dvě základní metody:

```
public static void Learn(){…}
public static void Classify(Evaluation evaluation){…}
```

V metodě Learn nalezneme algoritmus pro vyhodnocení reprezentanta. To získáme tak, že si projdeme jednotlivé množiny výsledků jedna, dva, tři, atd. a vypočteme průměrnou hodnotu reprezentanta. Zároveň získáme krajní body každé množiny. Získané hodnoty poté můžeme uložit do xml souboru anebo zahájit klasifikaci. Za zmínku stojí dříve zmiňovaná Euklidovská vzdálenost.

```
public double GetDistance(Evaluation actual)
{
  return Math.Sqrt(
         Math.Pow(Speed - actual.Speed, 2) +
         Math.Pow(Points - actual.Points, 2) +
         Math.Pow(Retuns - actual.Retuns, 2)
         );
}
```

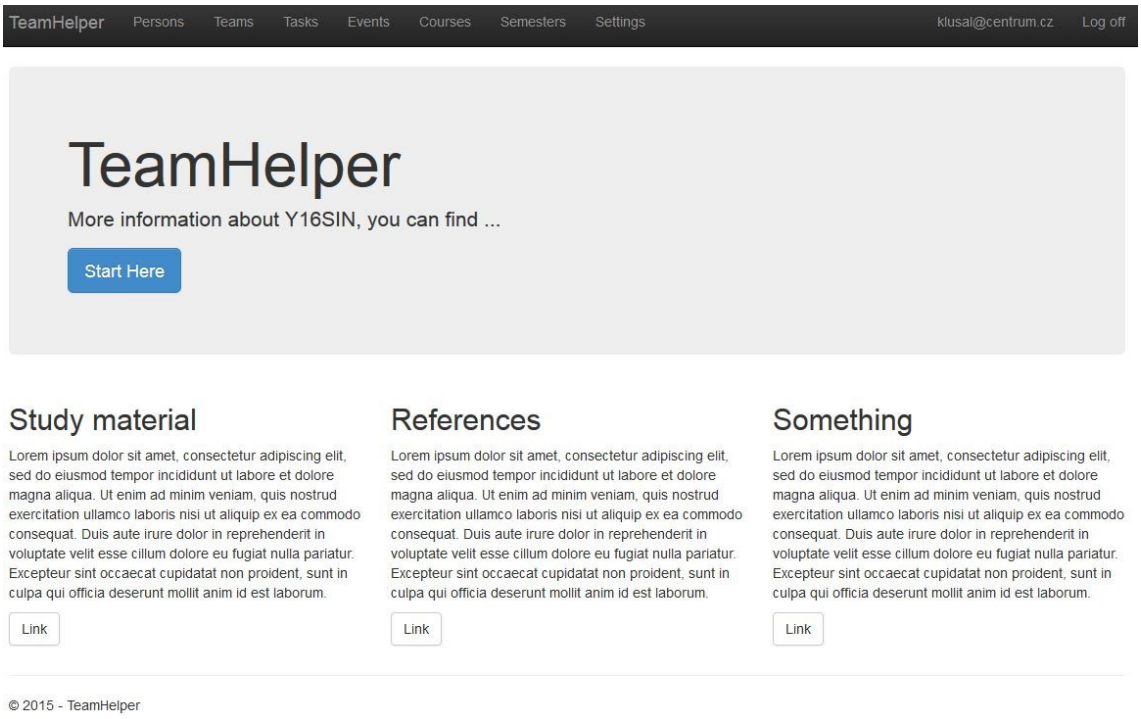
Metoda Classify slouží ke klasifikaci vstupu, u kterého potřebujeme provést vyhodnocení. Metoda zjistí, ke kterému reprezentantovi je nejblíže, a vyhodnotí výsledek. Tento výsledek je navržená známka a také navržené procentuální ohodnocení. Tyto hodnoty získáme díky výpočtu:

```
evaluation.Result = minWeight*minEvaluation.Result +
                    maxWeight*maxEvaluation.Result;
evaluation.ResultP = minWeight * minEvaluation.ResultP +
                     maxWeight * maxEvaluation.ResultP;
```

Kde minWeight a maxWeight představují vzdálenost vstupu od minimálního a maximálního ohodnocení. Výsledek a procentuální výsledek představuje Result a ResultPercentage. Díky tomuto vzorci jsme pak schopni jasněji říci, kolik student získal bodů.

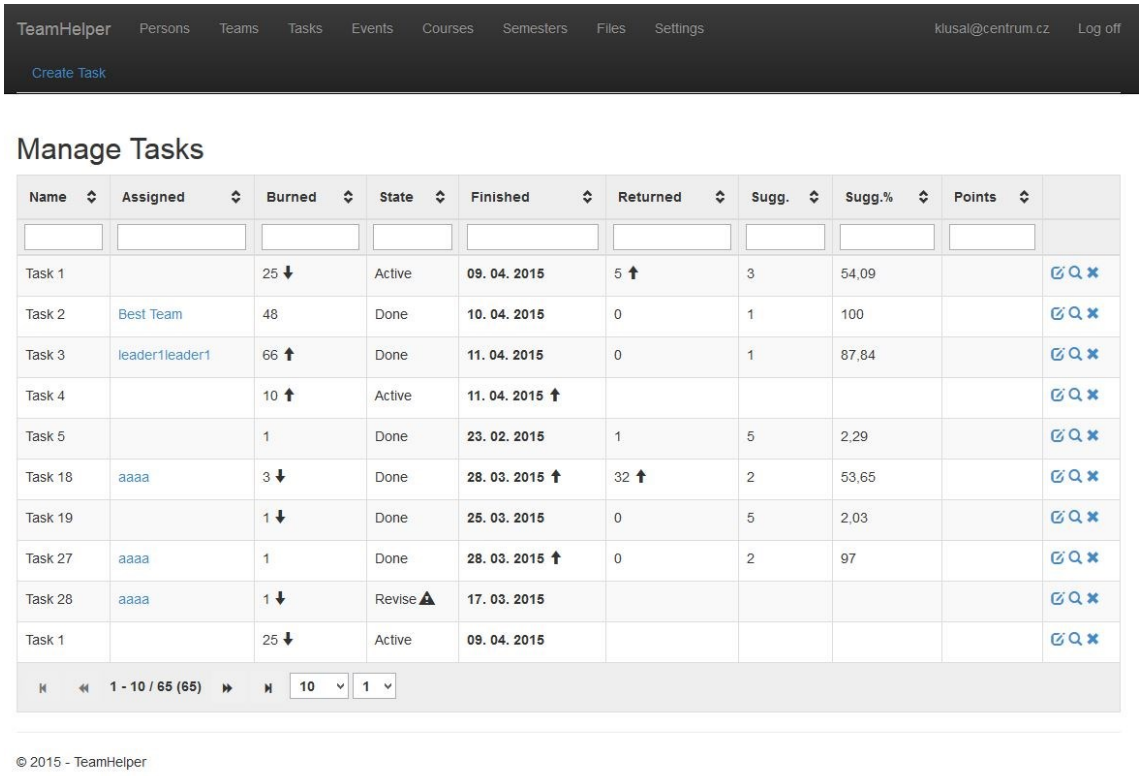
5.5 Vizualizace

Na prvním obrázku (Obrázek 5.4) lze vidět úvodní obrazovku, která uživatele přivítá na stránkách systému. Aplikace využívá framework Bootstrap a je tedy připravena i pro menší zařízení jako je tablet nebo mobilní telefon. Jak již bylo řečeno, systém je určen pro výuku studentů, a proto by na úvodní obrazovce měly být nejdůležitější informace o předmětu, materiálech a případné odkazy. Bootstrap famework je výborný nástroj pro tvorbu stránek, se kterým i člověk s ne-grafickými vjemy dokáže vytvořit webové stránky jednoduše

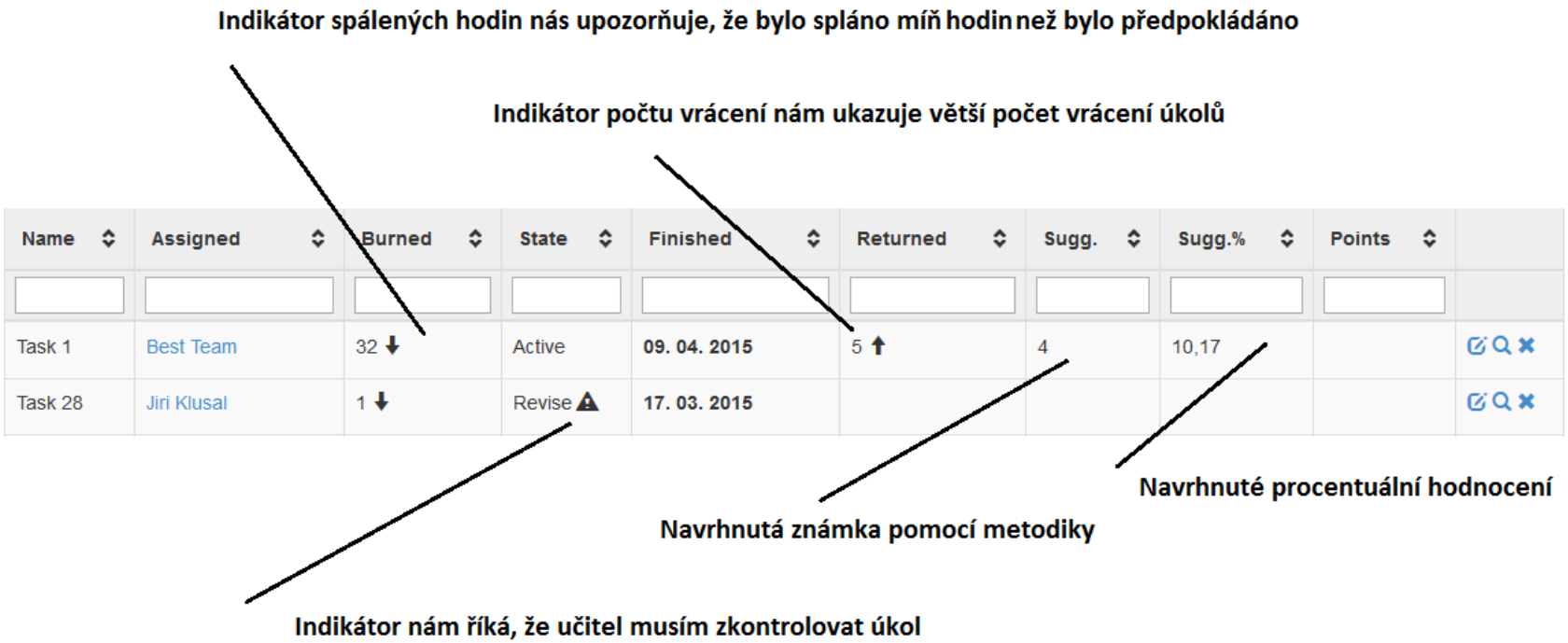


Obrázek 5.4: *Úvodní stránka*

Na obrázku 5.5 vidíme jednotlivé úkoly z pohledu administrátora systému. Indikátory napomáhají učiteli, aby si lépe všimnul, zdali je něco v nepořádku. Jde tedy vidět, že "Task 3", překročil počet přidělených hodin, a tedy spálil (v angličtině burned) více hodin, než mu bylo přiděleno. Jedná se pouze o ukazatel. Učitel je tedy upozorněn, že je něco v nepořádku, a může se na úkol podívat detailněji. Podobné indikátory jsou pak u ukončených úkolů (v angličtině finished) a vrácených (returned) úkolů. Vykřičník u stavu "revise", znamená, že by měl učitel úkol zkontrolovat a pak případně vrátit k přepracování nebo ohodnotit. Pokud úkol splňuje podmínky zpracování dat, je mu navrhnuta známka s procentuálním hodnocením. Toto hodnocení nalezne pak učitel ve sloupečku "Sugg." a "Sugg.%". Detailnější popisek a vysvětlivky jsou vidět na obrázku 5.6.

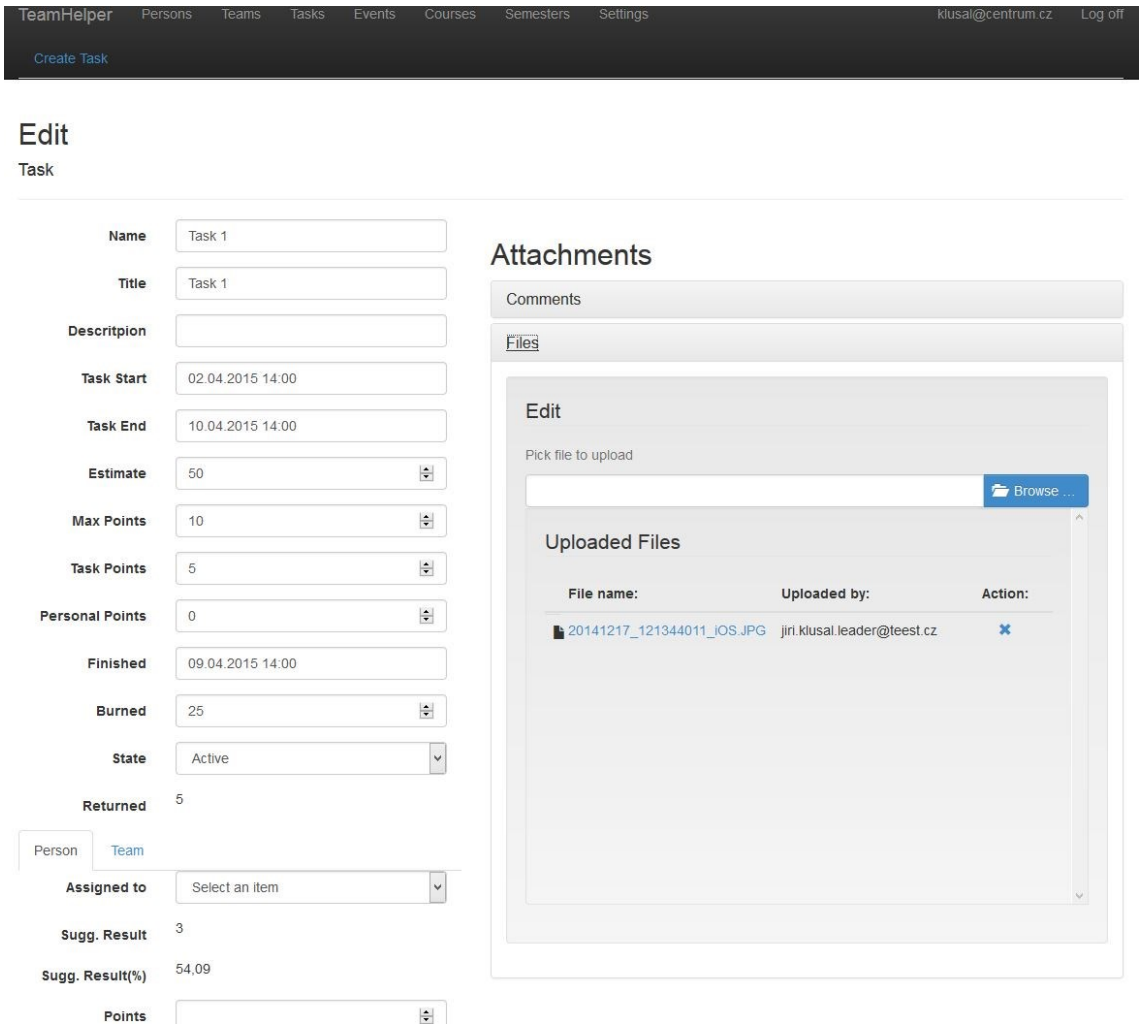


Obrázek 5.5: *Přehled úkolů s hodnocením a indikátory*



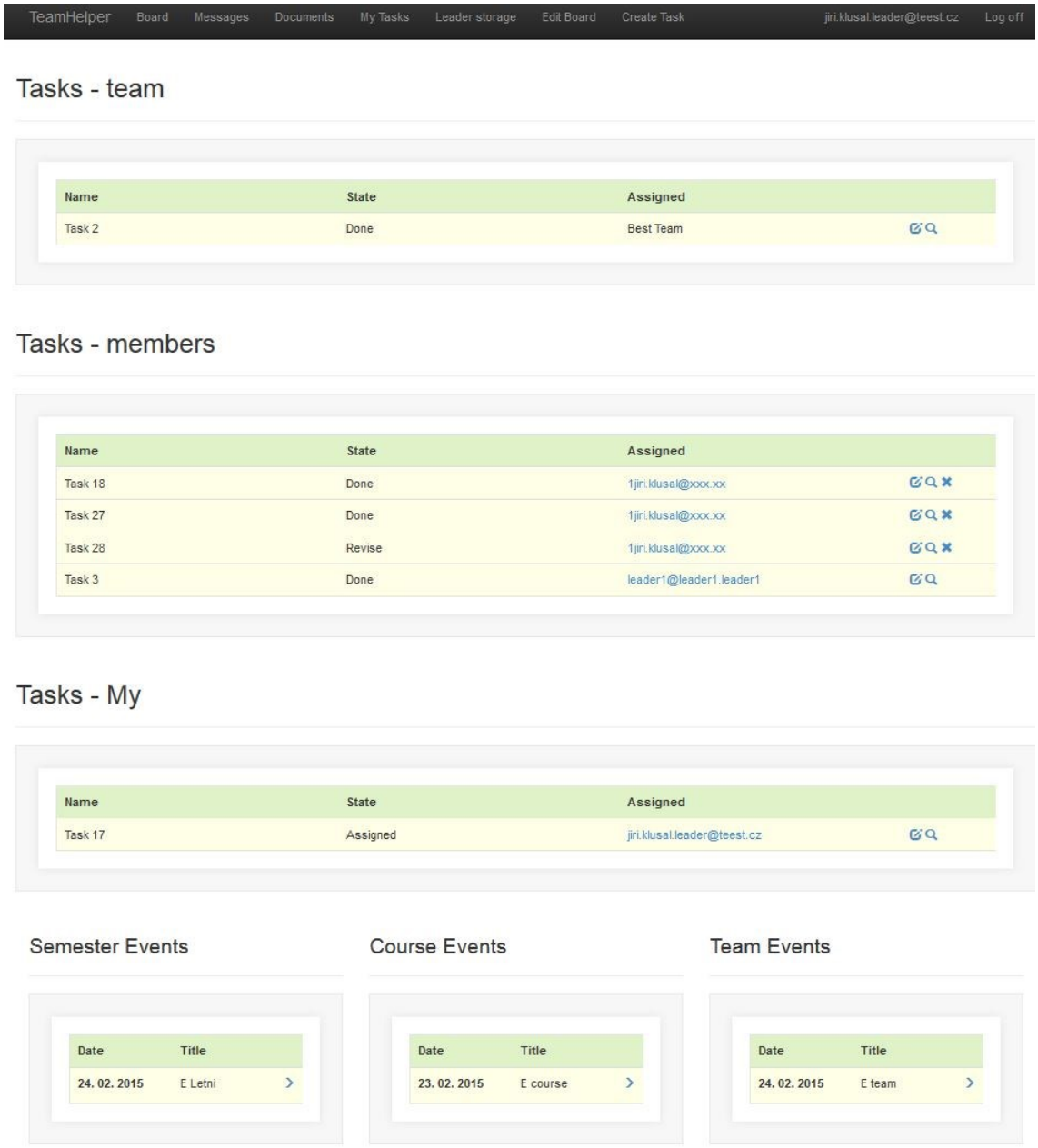
Obrázek 5.6: Přehled úkolu s indikátory a legendami

Ve chvíli, kdy učitele zaujmou ukazatele nebo případně potřebuje zkontrolovat studentovu práci, může vstoupit do editace úkolu. Učitel pak může upravovat úkol, ohodnotit práci, případně prodloužit termín. Jednotlivé hodnoty jsou vidět na obrázku 5.7. Pokud byl přiložen soubor, učitel si jej může stáhnout a projít. Soubor je dostupný i přímo na FTP serveru. Pokud student zanechal komentář, učitel si jej může přečíst, případně zanechat svůj komentář, když studentovi vrací práci na předělání.



Obrázek 5.7: *Editace úkolu*

Každý student, který je součástí týmu, má svou nástěnku, pomocí které vidí, co se děje. Student dokáže velice rychle zjistit (obrázek 5.8), jaké úkoly jsou přiřazeny celému týmu, co dělají ostatní členové týmu a jaké úkoly momentálně řeší. Mimo to na nástěnce vidí, jaké události se chystají v rámci semestru, kurzu (cvičení) a jaké v týmu. Součástí této stránky je i prostor, který zpravuje leader týmu, a díky CK editoru dokáže přizpůsobovat nástěnku k potřebám týmu. CK editor je HTML textový editor, pomocí kterého jednoduše přizpůsobujeme/měníme obsah stránek.



Obrázek 5.8: *Přehled úkolů a událostí*

5.6 Nasazení

Jak bylo zmíněno v nefunkčních požadavcích, aplikace pro svůj chod vyžaduje IIS (internet information service) a je psaná v jazyce C#., a to pro verzi .NET Framework 4.5. Aplikace využívá ASP. NET MVC 5. Tedy momentálně jediným řešením, jak nasadit aplikace, je mít k dispozici operační systém Windows. Aplikace pro uskladnění dat využívá MSSQL 2012.

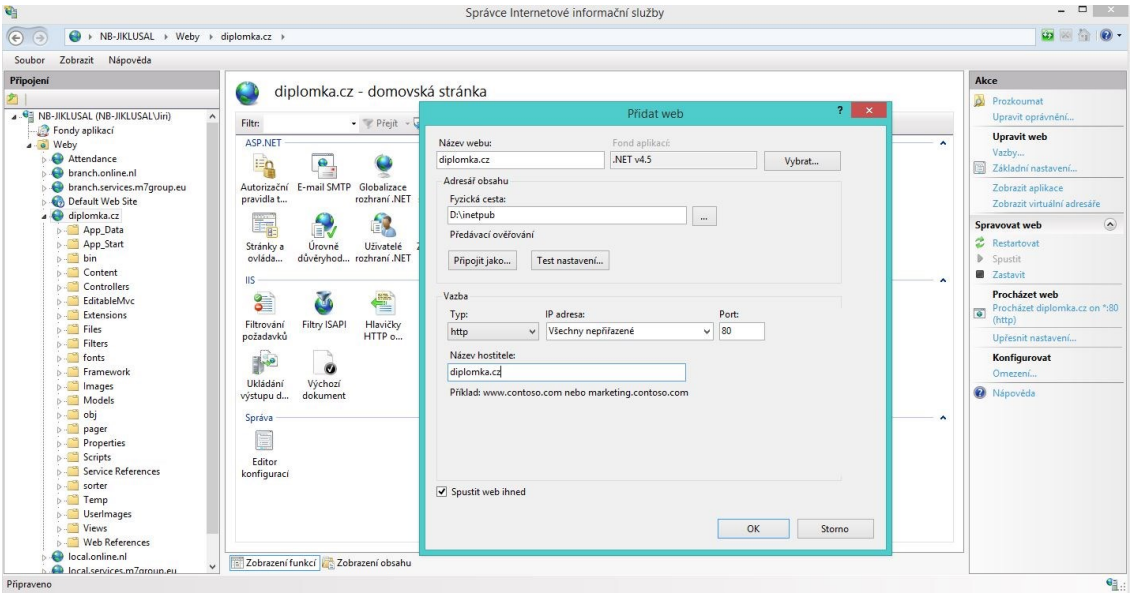
Věřím tomu, že pokud by bylo nutné aplikaci zprovoznit na Linuxech, bylo by možné aplikaci upravit tak, aby využívala MVC 4, které je momentálně v poslední beta verzi Mona podporováno, a tím případně i snížit cílený Framework.

Softwarové požadavky

- OS Windows – s IIS 8
- Framework .NET 4.5
- Databázový server MSSQL 2012

Konfigurace IIS

Na IIS je třeba vytvořit vlastní webovou alikaci, která poběží v aplikačním fondu .NET v4.5. Pro případné lokální testování doporučuji nastavit do hosts souboru: „127.0.0.1 diplomka.cz“



Obrázek 5.9: IIS konfigurace

Konfigurace databáze

Pro vytvoření databáze, je třeba spustit sql scripty, které jsou přiloženy k práci, a to v pořadí:

- CreateDatabaseSchema.sql – script pro vytvoření databáze
- InsertData.sql – script pro základních dat (enumy, admin uživatel)
- Task_trigger - script pro logování změny stavů úkolu

Přihlášení

Po splnění všech předešlých kroků bychom měli být schopni se nalogovat do systému. Uživatelské jméno pro přihlášení je: „admin@admin.admin“ a heslo je „admin“. Po přihlášení doporučuji změnit heslo, případně vytvořit vlastního uživatele s rolí admin.

6 Závěr

Jeden z hlavních cílů této práce bylo navrhnout a realizovat systém, který by pomohl studentům a učitelům při výuce. Systém nyní podporuje funkcionalitu, kterou mohou studenti využít při plnění úkolů a komunikace mezi sebou. Zároveň podporuje výuku a vytváří atmosféru srovnatelnou s reálným pracovním prostředím.

Docílil jsem tedy toho, že na jedné straně máme tým studentů, kteří řeší své úkoly, a na druhé straně učitele, který dokáže přehledně a efektivně jejich práci sledovat, starat se o jednotlivé studenty a cvičení. K tomu díky indikátorům výkonosti má přehled, zda student nevybočuje z krajních hodnot. V závěru mu poskytuje příležitost adekvátně posoudit výkon studenta a navrhnout odpovídající hodnocení.

Jednotlivé hodnoty, které nám reprezentují vstupy pro hodnocení, jako například čas strávený na úkolu nebo počet vrácení, umožní hodnocení úkolů i v praxi. Mohu tedy i v reálném pracovním prostředí pomoci vyhodnotit, jakým způsobem zaměstnanci plní pracovní úkoly. Jednotlivé studijní týmy stačí nahradit za projektové týmy, učitele za manažery. Navrhovaný systém lze využít například pro řízení menších projektů v rámci firmy. Získáme tím užitečný nástroj pro tracking issue managment s výhodami learning managment systému.

Systém je snadno rozšiřitelný o další moduly, jako je například modul autonomního vyhodnocování. Jako cestu do budoucna bych tedy viděl možnost rozšíření systému o modul MBTI, kde by byla také možnost vyhodnocení charakteru člověka, která by sloužila k přesnějšímu určování rolí jednotlivých uživatelů. Tyto role by v systému byly dále rozšířeny o to, čím se bude člověk v rámci projektu opravdu zabývat.

Pokud bych tedy vzal projektový tým zabývající se vývojem software, tak tím myslím programátora, testera, dokumentaristu, databázového experta, případně designera. Tím bychom získali další vstupní hodnotu, které bychom dokázali využít při skládání kvalitního týmu. V takovou chvíli můžeme vyhodnotit studenta, a na základě jeho výsledků při plnění úkolů zjistit, zdali mu úkoly typu programování šly lépe než testování, a tím složit ideální tým pro další semestr.

Pro mne byla práce velkým přínosem ze strany metodik pro vyhodnocování, kde jsem poprvé mohl vyzkoušet klasifikační algoritmy v praxi a dosáhnout tak automatického vyhodnocování. Za svůj největší úspěch považuji navrhnutí architektury pro tento systém, kde jsem se snažil spojit své dosavadní zkušenosti z praxe a ze školy, a navrhnout tak znovupoužitelnou architekturu, kterou můžu nasadit flexibilně na jiný projekt.

Použitá literatura

[1] ARLOW, Jim, NEUSTADT, Ila. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Brno: Computer Press, 2007. ISBN ISBN 978-802-5115-03

[2] DOLANSKÝ, Václav, MĚKOTA, Vladimír, NĚMEC, Vladimír. Projektový management. Praha: Grada Publishing, 1996. ISBN ISBN 80-716-9287-5

[3] SCHWALBE, Kathy. Řízení projektů v IT. Brno: Computer Press, 2007. ISBN 978-802-5115-268

[4] RICHTA, Karel a Jiří SOCHOR. Softwarové inženýrství I. Dot. 1. vyd. Praha: České vysoké učení technické, 1998, iv, 229 s. ISBN 80-01-01428-2

[5] HAJDIN, Tomáš. Agilní metodiky vývoje software. Brno, 2005, v, 71 s

[6] CEJTHAMR, Václav a Jiří DĚDINA. Management a organizační chování. 2., aktualiz. a rozš. vyd. Praha: Grada, c2010, 344 s. Expert (Grada). ISBN 978-80-247-3348-7

[7] BERKA, Petr. Dobývání znalostí z databází. Vyd. 1. Praha: Academia, 2003, 366 s. ISBN 80-200-1062-9

[8] Fasuga, Radoslav. Inteligentní metody ve vzdělávání. Ostrava: 2008

[9] managementmania. managementmania.com. [online]. 2011 [cit. 2015-04-18]. Dostupné z: https://managementmania.com

[10] ŠTAINER Consulting . O původu a vývoji typologie osobnosti MBTI. Typologie osobnosti. [online]. 2003 [cit. 2015-04-18]. Dostupné z: http://www.stainerconsulting.cz/mbti/Informace_o_osobnostnich_typech.php

[11] Info21, spol. s r.o.. Ukazatele KPI. Lighthouse21. [online]. [cit. 2015-04-18]. Dostupné z: http://www.lighthouse21.cz/ukazatele-kpi

[12] Ing. Jiří Stýblo. Řízení pracovního výkonu a výkonnosti z méně tradičních i tradičních pohledů. danarionline.cz/. [online]. 29.3.2007 [cit. 2015-04-18]. Dostupné z: http://www.danarionline.cz/archiv/dokument/doc-d1890v2568-rizeni-pracovniho-vykonu-a-vykonnosti-z-mene-tradicnich-i-tradicnich/

[13] Andrew Jensen. Measure an Applicant's Hard and Soft Skills to Hire Only the Best Employees. andrewjensen.net. [online]. 12.1.2011 [cit. 2015-04-18]. Dostupné z: http://www.andrewjensen.net/measure-job-applicants-hard-soft-skills/

[14] Hard skills a soft skills. chovani.eu. [online]. [cit. 2015-04-18]. Dostupné z: http://www.chovani.eu/hard-skills-a-soft-skills/c911http://www.w3schools.com/aspnet/mvc_intro.asp

[15] Tom Dykstra. Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application. asp.net. [online]. 30.7.2013 [cit. 2015-04-18]. Dostupné

z: http://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application

[16] Dependency Injection. http://nette.org/. [online]. 24. 9. 2012 [cit. 2015-04-18]. Dostupné z: http://doc.nette.org/cs/2.3/dependency-injection

[17] The Repository Pattern. msdn.microsoft.com. [online]. [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff649690.aspx

[18] About Moodle. moodle.org. [online]. [cit. 2015-04-18]. Dostupné z: https://docs.moodle.org/28/en/About_Moodle

[19] edmodo. edmodo.com. [online]. [cit. 2015-04-18]. Dostupné z: https://www.edmodo.com/

[20] Team Foundation Server. msdn.microsoft.com. [online]. [cit. 2015-04-18]. Dostupné z: https://msdn.microsoft.com/en-us/vstudio/ff637362.aspx

[21] k-Nearest Neighbors. statsoft.com. [online]. [cit. 2015-04-18]. Dostupné z: http://www.statsoft.com/textbook/k-nearest-neighbors

[22] Nearest Neighbors. http://scikit-learn.org. [online]. [cit. 2015-04-18]. Dostupné z: http://scikit-learn.org/stable/modules/neighbors.html

[23] Vincent Granville. Clustering idea for very large datasets. analyticbridge.com. [online]. 24.8.2013 [cit. 2015-04-18]. Dostupné z: http://www.analyticbridge.com/forum/topics/clustering-idea-for-very-large-datasets

[24] General Logistic Sigmoid Function . fxsolver.com. [online]. [cit. 2015-04-18]. Dostupné z: http://www.fxsolver.com/browse/formulas/General+Logistic+Sigmoid+Function

[25] Data Mining Preprocessing. mimuw.edu.pl. [online]. [cit. 2015-04-18]. Dostupné z: http://www.mimuw.edu.pl/~son/datamining/DM/4-preprocess.pdf

[26] HAN,Jiawei. Data mining concepts and techniques: Introduction. [online]. Urbana and Champaign: University of Illinois. 2006 [cit. 2015-04-18]. Dostupné z: http://134.208.3.165/course/2006/Fall/Data_mining/01.pdf

[27] Luka Crnkovic-Dodig. Fuzzy Math Part 1, The Theory. http://blog.peltarion.com/. [online]. [cit. 2015-04-18]. Dostupné z: http://blog.peltarion.com/2006/10/25/fuzzy-math-part-1-the-theory/

[28] Belbin. www.belbin.cz. [online]. [cit. 2015-04-20]. Dostupné z: http://www.belbin.cz/

[29] Budoucnost bude agilní. I pro projektové manažery. zework.wordpress.com. [online]. 24.6.2013 [cit. 2015-04-20]. Dostupné z: https://zework.wordpress.com/2014/06/20/budoucnost-bude-agilni-i-pro-projektove-manazery/

[30] Fahad Usmani. Functional-Organization-Structure. pmstudycircle.com. [online]. [cit. 2015-04-20]. Dostupné z: http://pmstudycircle.com/2012/08/what-is-a-functional-organization-structure/functional-organization-structure/

[31] Free Scrum Task Board PowerPoint Template. slidehunter.com. [online]. [cit. 2015-04-20]. Dostupné z: http://slidehunter.com/powerpoint-templates/scrum-task-board-powerpoint-template/

[32] Plánovací část projektu. xrol.sweb.cz. [online]. [cit. 2015-04-20]. Dostupné z: http://xrol.sweb.cz/plan.htm

[33] Yi Cao. Efficient K-Means Clustering. mathworks.com. [online]. 27.3.2008 [cit. 2015-04-20]. Dostupné z: http://www.mathworks.com/matlabcentral/fileexchange/19344-efficient-k-means-clustering-using-jit

[34] ASP.NET MVC. w3schools.com. [online]. [cit. 2015-04-20]. Dostupné z: http://www.w3schools.com/ASPnet/mvc_intro.asp

Seznam příloh

Součástí Diplomové práce je DVD.

Adresářová struktura přiloženého DVD:

- Dokumenty
- Enterprise Architect projekt
- Obrázky
- Projekt